E.U.I.T. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Internet Móvil

TÍTULO: LECCIONES PRÁCTICAS DE DESARROLLO DE APLICACIONES MÓVILES EN ANDROID

AUTOR: Adrián Rodríguez Castro

TUTOR: Antonio da Silva Fariña

DEPARTAMENTO: DIATEL

Miembros del Tribunal Calificador:

PRESIDENTE:	Marta Sánchez Agudo	
VOCAL:	Antonio da Silva Fariña	**Vº Bº.**
VOCAL SECRETARIO:	Jesús Moreno Blázquez	
DIRECTOR:		
Fecha de lectura:		
Calificación:		**El Secretario,**

RESUMEN DEL PROYECTO:

Los dispositivos móviles hoy en día son mucho más que un terminal con servicios de telefonía. Proporcionan multitud de servicios y funciones que a través de aplicaciones pueden ser explotadas de diferentes maneras. Muchos de estos terminales son complejos sistemas con procesadores de varios núcleos, gigabytes de almacenamiento, pantallas con resoluciones de alta definición y numerosos sensores que ofrecen exorbitantes posibilidades a los desarrolladores de aplicaciones.

El objetivo del PFC es principalmente docente. Se trata de proponer una batería de lecciones prácticas incrementales que cubran los objetivos docentes de la asignatura "Desarrollo de Aplicaciones Móviles". Con ellas se tratará de cubrir la mayoría de los aspectos prácticos de programación de que un desarrollador en Android puede encontrarse en un ámbito profesional real.

Escuela Universitaria de Ingeniería Técnica de Telecomunicación

Proyecto Fin de Carrera

LECCIONES PRÁCTICAS DE DESARROLLO DE APLICACIONES MÓVILES EN ANDROID

Proyecto fin de carrera. Adrián Rodríguez Castro

Tutor: Antonio da Silva Fariña

AGRADECIMIENTOS

He de expresar mi profundo agradecimiento a todas aquellas personas que me han dado la oportunidad de desarrollarme, tanto intelectual como profesional y personalmente. Su influencia ha sido de una importancia vital para poder llegar al punto en el que me encuentro, finalizando mi carrera universitaria.

No puedo nombrar a todos, pero si quiero reconocer específicamente el valor a algunos de ellos:

Mi familia, por su apoyo y empuje incondicional.

Mis compañeros de Universidad con los que tantas horas de estudio y esfuerzo pasé para aprobar todas las asignaturas. Gracias a ellos ha sido posible que yo presente hoy este documento.

Antonio da Silva, mi tutor, que me ha ayudado y guiado en la realización del proyecto para darle el carácter didáctico que tiene.

Los profesores de la universidad, que me dieron la base y conocimientos necesarios para la realización de este proyecto.

Las empresas de movilidad que me brindaron la oportunidad de convertirme en un profesional de este campo.

Y a todos aquellos que olvido nombrar.

"La imaginación es más importante que el conocimiento. El conocimiento se limita a todo lo que ahora conocemos y comprendemos, mientras que la imaginación abarca el mundo entero, todo lo que en el futuro se conocerá y entenderá."

Albert Einstein

Índice de contenidos

ILUSTRACIONES

TABLAS

CÓDIGO

RESUMEN

Los avances que se han producido en los últimos años en cuanto a potencia y capacidades de los teléfonos móviles que usamos de manera cotidiana, traen de la mano un auge en la demanda de aplicaciones de todo ámbito: desde aplicaciones generales de consumo, pasando por juegos, hasta aplicaciones que ofrecen soluciones internas a empresas.

Existen diferentes sistemas operativos para teléfonos móviles como se explicará más adelante en el capítulo introductorio. En dicho capítulo se da la justificación de por qué en el presente Proyecto Fin de Carrera se centra en el estudio del sistema operativo Android.

Primeramente se dará una visión global del estado del arte en cuanto al mundo de aplicaciones móviles se refiere. Se explicarán los pros y contras de cada sistema operativo, detallando el lenguaje de programación utilizado en cada uno de ellos y sus principales características.

Después, en el capítulo tres se estudiará con más profundidad el sistema operativo Android, desde su historia y orígenes, hasta los componentes básicos para la creación de una aplicación, pasando por la arquitectura interna del sistema o su máquina virtual.

Con esto se pretende que el lector tenga un contexto que le permita comprender los siguientes capítulos, que es donde está el núcleo de este Proyecto Fin de Carrera.

El cuarto capítulo trata de una serie de prácticas incrementales, que cubren una gran parte de las posibilidades que ofrece el sistema operativo Android para el desarrollo de aplicaciones. Se ha pretendido que la dificultad vaya de menos a más y que las prácticas se vayan apoyando en las anteriores, para tener al final una única solución que englobe todas las lecciones.

El último capítulo quiere englobar el uso de todas las lecciones aprendidas en las lecciones anteriores para crear una aplicación que bien podría ser una aplicación real para un cliente. Se trata de una aplicación que muestra en tiempo real información sobre las cámaras de tráfico de la ciudad de Madrid.

ABSTRACT

The improvements that have occurred in recent years in terms of power and capabilities of mobile phones that we use on a daily basis, bring an increment in demand for all kind of applications, from general consumer applications, games or even internal applications that offer solutions to companies.

There are different operating systems for mobile phones as will be explained later in the introductory chapter. In that chapter the answer for why this Thesis focuses on the study of the Android operating system is given as well.

First an overview of the state of the art about the world of mobile applications will be referred. The pros and cons of each operating system will be explained, detailing the programming language used in each of them and their main characteristics.

Then in chapter three will be discussed in more depth the Android operating system, from its history and beginnings to the main components for the creation of an application, to the internal architecture of the system or virtual machine.

The goal of chapter three is to give the readers a context that allows them to understand the following chapters, where the core of this Thesis is.

The fourth chapter contains a series of incremental practices covering a large part of the potential of the Android operating system for application development. Those practices grow in difficulty and are supported by the previous in order to have at the end a single solution that fits all lessons.

The last chapter wants to embrace the use of all the lessons learned in previous lessons to create an application that could well be an actual application for a client. It is an application that displays real-time information off traffic cameras of the city of Madrid.

CAPÍTULO 1. INTRODUCCIÓN

Este capítulo es el punto de entrada al resto del documento y en él se puede encontrar una visión general sobre el tema que nos ocupa así como los objetivos y motivaciones para la realización del presente proyecto.

1.1 Visión general

Los dispositivos móviles hoy en día son mucho más que un terminal con servicios de telefonía. Nos proporcionan multitud de servicios y funciones que a través de aplicaciones podemos explotar de diferentes maneras. Muchos de estos terminales son complejos sistemas con procesadores de varios núcleos, gigabytes de almacenamiento, pantallas con resoluciones de alta definición y numerosos sensores que ofrecen exorbitantes posibilidades a los desarrolladores de aplicaciones.

España se ha posicionado como el país líder en uso de *smartphones*, con un 66% de penetración, por delante de Inglaterra, Francia o Alemania, entre otros, según el Centro Universitario de Tecnología y Arte Digital U-Tad, citando datos del informe *Spain Digital Future in Focus*, de Comscore (Europa Press, 2013).

En este estudio se observa que España es el país con mayor proporción de smartphones con sistema Android en el mercado, con un 92% de la cuota. Por detrás de esta plataforma se posicionan iOS de Apple (4,2%), Windows (1,9%), Symbian (0,9%) y Blackberry (0,1%).

A nivel mundial Android ha alcanzado el 64,6% de la cuota de smartphones, lo que convierte a la plataforma de Google en la primera en ventas de teléfonos inteligentes. Está tónica se mantiene en Europa, donde el 70,4% de los terminales usan este sistema operativo.

Las aplicaciones para la plataforma Android se desarrollan principalmente en Java. Según el índice TIOBE, este lenguaje sigue estable en prácticamente un 16% de popularidad entre programadores (TIOBE). Hay que recordar que, mientras las aplicaciones de alto nivel se desarrollan en el lenguaje de Oracle, siempre se puede recurrir a C para realizar librerías a bajo nivel, usando para ello el Native Development Kit.

Debido a esto se ha considerado que el desarrollo de una aplicación móvil para este sistema es un tema interesante, complejo y con una alta relevancia en el presente y en un futuro próximo. Su carácter de libre distribución, el hecho de que profesionalmente me dedico a desarrollar aplicaciones para ésta y otras plataformas y el crecimiento anteriormente citado han sido determinantes en la elección de la temática.

Para concluir con esta visión general, he de añadir que las pretensiones de este proyecto no terminan con la implementación del prototipo. En la consecución del proyecto se intentará establecer unas líneas guía de programación de aplicaciones para dispositivos móviles.

1.2 Objetivos

El presente proyecto fin de carrera tiene un carácter principalmente pedagógico y se basa en una serie de lecciones prácticas incrementales. Con ellas se tratarán de cubrir todos los ámbitos que un desarrollador de Android puede encontrarse en un ámbito profesional. Las lecciones terminan en una aplicación que abarca lo aprendido en todas las prácticas. Se trata de una aplicación en la que se muestra información visual geoposicionada de las cámaras de tráfico del Ayuntamiento de Madrid. Para ello se usarán datos públicos reales obtenidos en tiempo de ejecución. Es una

aplicación que presenta la problemática real en una aplicación comercial: desde la descarga y procesado a la presentación de los datos, pasando por las interacciones permitidas al usuario.

La aplicación pretende ser una prueba de concepto de una aplicación real, y es interesante remarcar sus posibilidades de mejora. Como posible añadido se podría permitir al usuario marcar cámaras favoritas y mostrarlas en la pantalla principal, o calcular rutas alternativas de circulación si el usuario ve que una zona está congestionada.

La aplicación deberá ser lo más modular posible para que sea fácilmente escalable a otras fuentes de datos diferentes y poder así implantarla en distintos entornos. Esto es posible gracias al conocimiento de la arquitectura, componentes básicos, metodologías y modelos usados en la plataforma. Esto será estudiado en el capítulo tres a modo de presentación.

CAPÍTULO 2. ESTADO DEL ARTE

El propósito de este capítulo es situar el proyecto dentro de un ámbito más amplio en las tecnologías móviles.

2.1 Dispositivos móviles

En la actualidad, el uso de dispositivos móviles se ha convertido en un estándar para la comunicación en prácticamente cualquier área de nuestra sociedad, la cual ha experimentado un cambio muy importante en este aspecto.

Los dispositivos móviles son aparatos de tamaño reducido que facilita su transporte, con capacidad de cómputo y conexión a redes que han sido fabricados para llevar a cabo una funcionalidad determinada. Dentro de esta categoría se pueden incluir entre otros, ordenadores portátiles, reproductores de música y video, PDAs, videoconsolas y por supuesto teléfonos móviles.

Tratando de acotar un poco la definición de dispositivo móvil para acercarnos al tema que nos ocupa debemos centrarnos en los teléfonos móviles, cuya funcionalidad básica hasta hace poco tiempo ha sido la comunicación por voz a través de una red además del envío de mensajes cortos de texto. Sin embargo, conducidos por los cambios sociales comentados anteriormente, estos servicios se han visto enormemente ampliados en un corto periodo de tiempo.

Acceso a internet, cámaras de alta resolución, conexión WiFi, pantallas táctiles, múltiples sensores o localización por GPS se han convertido necesariamente en propiedades básicas de estos dispositivos, y con ellas, el desarrollo de aplicaciones que explotan dichas características.

Este auge de la tecnología móvil ha propiciado gran diversidad de opciones en lo que a sistemas operativos se refiere, y es por eso que la elección a la hora de desarrollar para alguna de estas plataformas es de gran importancia. Como se ha comentado anteriormente en la introducción, atendiendo a datos objetivos, el predominio de Android es claro.

2.2 Sistemas operativos para dispositivos móviles

Para la realización del presente proyecto, se ha utilizado el sistema operativo Android cuyas características serán convenientemente explicadas en el capítulo tres. En este apartado hablaremos del resto de alternativas existentes en el mercado que fueron valoradas antes realizar una elección.

• 2.2.1 iOS

iOS7 es la última versión del aclamado sistema operativo utilizado por Apple en el iPhone. La sencillez y facilidad de uso han estado siempre ligadas a este sistema ofreciendo a los usuarios una experiencia agradable e intuitiva. En el ámbito de los terminales móviles, el iPhone en cualquiera de sus versiones se encuentra muy asentado entre el público por lo que un desarrollo en esta plataforma tendría un gran alcance entre la sociedad.

Es un sistema operativo basado en BSD Mach 3, y como tal, se pueden usar todas las librerías estándar de Unix y C.

Aunque a priori pareciera que las características de este sistema operativo son adecuadas para iniciar un proyecto, es justo a la hora de desarrollar cuando encontramos las principales trabas. El SDK está disponible para su descarga, pero es necesario tener un Mac para que este funcione y estar registrado en la web de desarrolladores de Apple. Por otro lado, aun cumpliendo el requisito anterior, solo se pueden probar aplicaciones en el emulador que viene con el SDK. Para probarlas en un iPhone o subirlas al App Store para su distribución, habría que pagar una licencia de 99$.

El lenguaje utilizado para desarrollar aplicaciones para iOS es Objective C. Es un lenguaje que nació a principios de los años ochenta, y hereda su sintaxis de SmallTalk (Lount, 2004) de Xerox. La sintaxis es diferente a Java o a C# y suele causar dificultades a la hora de seguir el flujo de acciones de un programa la primera vez que se ve. He aquí un ejemplo simple de invocación de método para entender esto:

- Java → `this.drawCircle(100, 100, 25, RED, BLUE);`
- Objective C → `[self drawCircleAtCenterX:100.0 andY:100.0 withRadius:25.0 withStrokeColor:[UIColor redColor] filledWithColor:[UIColor blueColor]];`

Puede llegar a ser más intuitivo, ya que intenta simular una frase construida en inglés, pero de primeras supone un reto.

Los inconvenientes aquí expuestos son suficientes para descartar esta opción.

Ilustración 1. Logotipo iOS7

• 2.2.2 BlackBerry OS

RIM (Research In Motion) fue uno de los pioneros en la integración del correo electrónico en un terminal móvil, el BlackBerry. El sistema operativo está claramente orientado al uso profesional como gestor de agenda y correo electrónico, también cabe destacar el teclado físico de tipo QWERTY que es seña de identidad en todos los terminales del fabricante.

Tras una pérdida de cuota de mercado muy acusada, RIM ha intentado reengancharse a las tendencias actuales con el sistema operativo BlackBerry 10. Es un SO basado en QNX, que fue adquirido por BlackBerry en 2010, y ofrece gestos, capacidades multitarea, y mejoras multimedia sin olvidar el núcleo que hizo fuerte a BlackBerry tan fuerte en el sector profesional: la facilidad para gestionar cuentas de correo, contactos y calendario.

Esta última versión incluye una capa de ejecución de aplicaciones Android. Actualmente solo es capaz de lanzar aplicaciones compiladas con la versión 2.3 de este sistema operativo

(Gingerbread), y es una versión bastante obsoleta y con cada vez menos presencia entre los terminales (18% de terminales conectados al Google Play) (Google Inc., 2014).

BlackBerry OS es un sistema que permite a desarrolladores independientes programar aplicaciones, pero en el caso de que éstas requieran tener acceso a ciertas funcionalidades restringidas deberán estar firmadas digitalmente con un certificado asociado a una cuenta de desarrollador de RIM. El motivo por el que se desechó esta opción fue por el enfoque claramente profesional que tienen estos dispositivos y que no concuerdan con las aspiraciones del presente proyecto.

Ilustración 2. Logotipo BlackBerry 10

• 2.2.3 Windows Phone 8

La última versión el sistema operativo de Microsoft para dispositivos móviles es Windows Phone 8. Es la primera versión que deja atrás el núcleo de Windows CE y lo reemplaza con Windows NT, con muchos componentes compartidos con la versión de escritorio, Windows 8.

Esta versión del sistema operativo introduce una interfaz diferente que presenta *mosaicos dinámicos* que se actualizan constantemente en función de las aplicaciones que el usuario utiliza. Otro de los nuevos conceptos son los hub, que agrupan funcionalidades y aplicaciones referentes a un mismo tema mejorando así la usabilidad del sistema.

De cara al desarrollo, una de las ventajas que ofrece esta plataforma es la posibilidad de desarrollar en varios lenguajes de programación diferentes como C#, VisualBasic o Javascript + HTML5, por lo que programadores de otros ámbitos pueden abordar la movilidad desde la seguridad de un lenguaje conocido.

La principal razón por lo que he descartado el desarrollo del prototipo para Windows 8 fue la cuota de mercado que, aunque emergente, es aún mucho inferior a la de Android.

Ilustración 3. Logotipo Windows 8 Phone

2.3 Java 2 Micro Edition

La empresa Sun Microsystems lanzó a mediados de los años 90 el lenguaje de programación orientado a objetos Java. Debido a su gran robustez y sobre todo a la independencia en cuanto a ejecución en distintas plataformas, este nuevo lenguaje se convirtió en uno de los más populares a la hora de desarrollar aplicaciones de todo tipo. Esta portabilidad que lo hizo tan importante fue debido a la famosa máquina virtual Java o JVM, ya que al compilar un fichero fuente en Java no se genera directamente un ejecutable binario, se genera el llamado *bytecode* de Java que más tarde es traducido por la JVM instalada en la plataforma, para las cuales existe una máquina virtual Java distinta, ya sea en un ordenador, un dispositivo móvil o incluso electrodomésticos. La JVM conoce el conjunto de instrucciones propias de la plataforma y es capaz de traducir el *bytecode* en código nativo.

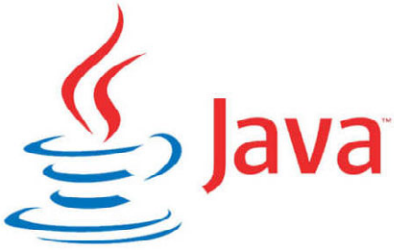


Ilustración 4. Logotipo Java

La edición Java 2 Micro Edition fue presentada por Sun Microsystems en 1999 y nació con el propósito de habilitar aplicaciones Java para pequeños dispositivos con capacidades restringidas tanto en pantalla gráfica como de procesamiento y memoria ya que las necesidades de los usuarios de telefonía móvil han cambiado radicalmente en los últimos años demandando cada vez más servicios y prestaciones tanto en hardware como en software. Viendo lo rápido que se ha asentado esta tecnología, podríamos decir que J2ME es el futuro para la industria móvil.

Debido a la gran amplitud existente en el mercado de las tecnologías de la información, Sun Microsystems se vio obligada a ofrecer soluciones personalizadas según el ámbito tecnológico en cuestión y de acuerdo a las necesidades de cada uno, de ahí nacieron distintas versiones de Java 2 que se han ido ampliando con el tiempo. A continuación veremos las principales versiones que existen.

Java Platform, Enterprise Edition (J2EE)

Es la versión orientada al entorno empresarial, por lo que no está pensado para ejecutarse sobre un solo equipo, sino para hacerlo sobre una red de ordenadores de manera distribuida y remota.

Java Platform, Standard Edition (J2SE)

Está orientada al desarrollo con independencia de la plataforma al generar el bytecode de Java y ejecutarlo en la plataforma destino a través de la JVM. Esta versión contiene un conjunto básico de las APIs de Java orientadas a la programación de aplicaciones para el usuario final.

Java Platform, Micro Edition (J2ME)

La J2ME es la versión enfocada a la aplicación de la tecnología en pequeños dispositivos con capacidades computacionales y gráficas restringidas. Para algunos de estos dispositivos se utiliza una máquina virtual distinta, KVM (Kilo Virtual Machine, debido a que solo necesita unos pocos Kilobytes para ejecutar) además de incluir un recolector de basura que optimiza el uso de memoria.

Java Card

Esta versión está enfocada a la ejecución de pequeñas aplicaciones Java en tarjetas inteligentes y dispositivos similares de modo que la tarjeta tenga una funcionalidad en un dominio específico. Esta tecnología es ampliamente utilizada en las tarjetas SIM así como en las tarjetas de monedero electrónico.

Todas las ediciones comparten un conjunto de las APIs básicas de Java que encontramos principalmente en los paquetes `java.lang` y `java.io`. A partir de esta información, se puede concluir que JME es un subconjunto de JSE, al igual que este lo es de JEE como se puede observar en la siguiente ilustración.

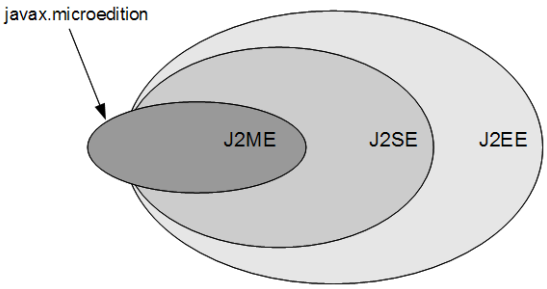


Ilustración 5. Conjuntos de versiones de Java

Sin embargo, existe una pequeña peculiaridad respecto a J2ME, y es que no es realmente un subconjunto al completo, ya que al estar enfocado a dispositivos de capacidades restringidas nos encontramos con la inclusión entre sus paquetes de `javax.microedition` como se puede observar en la figura anterior. Por lo tanto llegamos a la conclusión de que Java 2 Micro Edition es una versión muy específica del lenguaje Java que ha sido creada para desarrollar, instalar y ejecutar software los ya nombrados dispositivos móviles de capacidad reducidad.

• 2.3.1 Arquitectura

La arquitectura de J2ME define configuraciones, perfiles y paquetes opcionales como elementos básicos para desarrollar aplicaciones que según la plataforma de destino, se combinan consiguiendo optimizar los resultados.

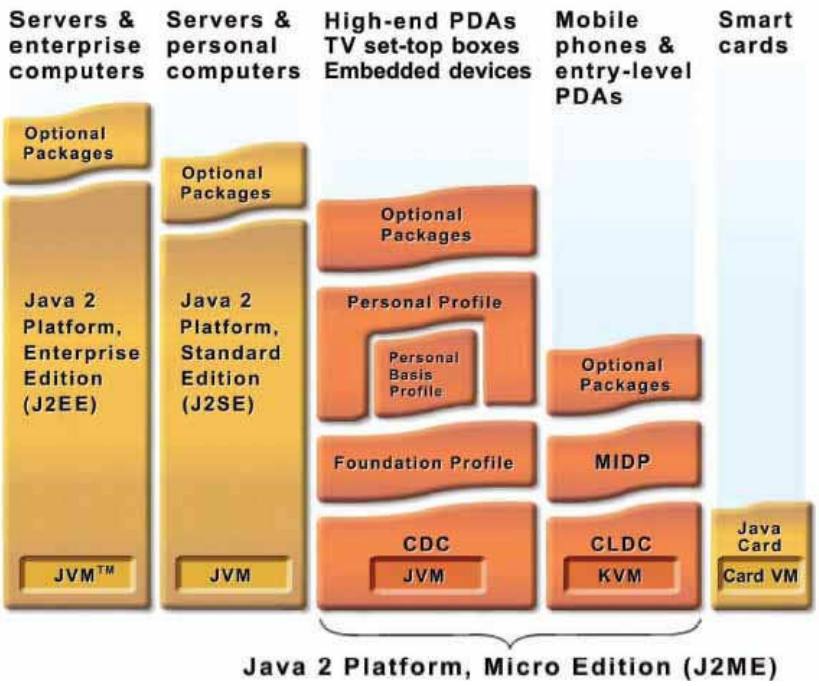


Ilustración 6. Arquitectura de la plataforma Java 2

En la figura anterior se puede observar la arquitectura de Java 2 al completo, y en concreto como la plataforma J2ME se subdivide en 2 grandes bloques con pequeñas diferencias basadas en los elementos definitorios comentados anteriormente y que serán explicados a continuación.

Máquinas virtuales

Existen dos tipos de máquina virtual para utilizar con Java Micro Edition, cada una tiene distintos requisitos y está pensada para su uso diferentes tipos de dispositivos de reducido tamaño.

KVM es la primera de ellas y se trata de una implementación de máquina virtual reducida (de hecho, es la JVM más pequeña desarrollada por Sun). Está orientada a dispositivos con bajas capacidades computacionales y de memoria, está escrita en C y estas son algunas de sus características principales

- Pequeña carga de memoria (entre 40 y 80 Kb)
- Alta portabilidad
- Modular

Sin embargo, esta baja ocupación de memoria genera ciertas limitaciones con respecto a la JVM clásica. Estas limitaciones son las siguientes:

- No hay soporte para tipos en coma flotante. Los dispositivos carecen del hardware necesario
- No existe soporte para JNI (*Java Native Interface*)
- Solo existen los cargadores de clases predeterminados
- No es posible finalizar las clases mediante `Object.finalize()`
- El manejo de excepciones es limitado

CVM, o *Compact Virtual Machine* soporta las mismas características que la JVM de Java SE, y está pensada para dispositivos con procesadores de 32 bits y más de 2 Mb de memoria RAM. Estas son sus principales características:

- Sistema de memoria avanzado
- Recolector de basura ágil y modular
- Soporte nativo de hilos
- Portabilidad y rápida sincronización
- Baja ocupación de memoria

Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos, y es en estas APIs donde se describen las particularidades comunes a la plataforma objeto del desarrollo. Existen 2 configuraciones en J2ME:

CLDC (*Connected Limited Device Configuration*) está orientada dispositivos dotados de conexión y con limitaciones de cómputo, memoria y capacidad gráfica que vienen dadas por el uso de la KVM. Los dispositivos que usen esta configuración deben cumplir las siguientes exigencias:

- Disponer entre 160 y 512 Kb de memoria no volátil
- Procesador de 32 bits y un mínimo de 25 Mhz
- Ofrecer bajo consumo al operar habitualmente mediante baterías
- Disfrutar de conexión a una red inalámbrica

Además, la CLDC aporta las librerías mostradas en la tabla (Subconjuntos todas ellas de J2SE) que ofrecen algunas funcionalidades extra como el soporte de E/S, acceso a redes y seguridad básica.

Paquete	Descripción
Java.io	Clases y paquetes estándar de E/S
Java.lang	Clases e interfaces de la máquina virtual
Java.util	Clases, interfaces y utilidades estándar
Javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 1. Librerías de configuración CLDC

CDC (*Connected Device Configuration*) está pensada para dispositivos con más recursos como podrían ser decodificadores de televisión digital, sistemas de navegación de automóviles, electrodomésticos o televisores con internet que deben cumplir estas recomendaciones:

- Procesador de 32 bits
- Más de 2 Mb de memoria total (entre RAM y ROM)
- Conectividad a algún tipo de red

De igual manera que en la configuración anterior, en la siguiente tabla se muestran las librerías incluidas en este tipo de configuración.

Paquete	Descripción
Java.io	Clases y paquetes estándar de E/S
Java.lang	Clases e interfaces de la máquina virtual
Java.lang.ref	Clases de referencia
Java.lang.reflect	Clases e interfaces de reflection
Java.math	Paquete matemático
Java.net	Clases e interfaces de red
Java.security	Clases e interfaces de seguridad
Java.security.cert	Clases de certificados de seguridad
Java.text	Paquete de texto
Java.util	Clases, interfaces y utilidades estándar
Java.util.jar	Utilidades de archivos Jar
Java.util.zip	Utilidades para archivos ZIP y comprimidos
Javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 2. Librerías de configuración CDC

Perfiles

Un perfil es una agrupación de APIs que definen el modelo de ciclo de vida de la aplicación, la IU y el acceso a propiedades concretas del dispositivo. Para ello identifican conjuntos de dispositivos cuyo nexo es la funcionalidad que proporcionan y el tipo de software que se ejecutará en ellos. Igualmente que en la selección de máquina virtual según la configuración, existen ciertos perfiles para usar sobre CDC como son el Foundation Profile o el Persona Profile y otros distintos para CLDC que son el MID Profile y el PDA Profile.

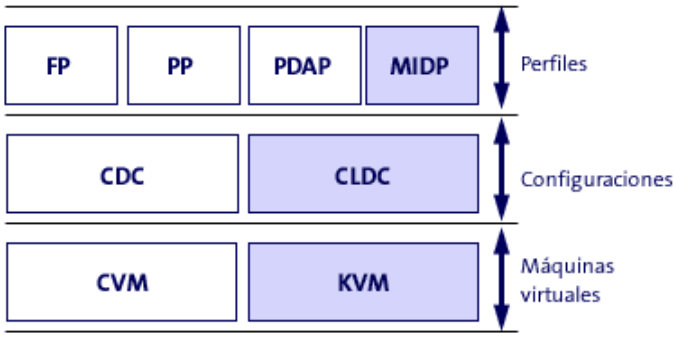


Ilustración 7. Perfiles Java ME

• 2.3.2 MIDlets

Las aplicaciones en Java 2 Micro Edition que se realizan utilizando el perfil MIDP reciben el nombre común de MIDlets, por consiguiente, y siguiendo el esquema mostrado en la figura anterior se define un MIDlet como una aplicación JME realizada con el perfil MIDP, sobre una configuración CLDC y que utiliza la máquina KVM. Partiendo de esta afirmación y de las características que presentan los dispositivos móviles sobre los que corre Android, para alcanzar el objetivo del presente proyecto, el perfil MIDP es el único que por definición podremos utilizar.

Este perfil en concreto incluye las siguientes librerías mostradas en la tabla, las cuales pueden ser utilizadas por el programador de la aplicación en cualquier momento.

Paquete	Descripción
Java.io	Clases y paquetes estándar de E/S
Java.lang	Clases e interfaces de la máquina virtual
Java.util	Clases, interfaces y utilidades estándar
Javax.microedition.util	Marco de ejecución para aplicaciones
Javax.microedition.lcdui	Interfaces de usuario
Javax.microedition.rms	Almacenamiento persistente en el dispositivo
Javax.microedition.io	Clases e interfaces de conexión genérica CLDC

Tabla 3. Librerías de los MIDlets

Existen solo 3 estados posibles durante el ciclo de vida de un MIDlet: *Pausado*, *Activo* o *Destruido*, pudiendo estar solo en un estado a la vez. En la siguiente figura se muestra cómo es posible pasar de un estado a otro.

CAPÍTULO 3. SISTEMA OPERATIVO ANDROID

En este capítulo se estudiará a fondo este sistema operativo para conocer sus principales características.

3.1 Introducción

Android es el sistema operativo de Google orientado a dispositivos móviles. Anunciado en Noviembre de 2007 (Open Handset Alliance, 2007), fue finalmente lanzado al mercado en su versión 1.0 en octubre de 2008, con la venta del HTC Dream, también conocido como T-Mobile G1 (GSMArena, 2008). Está basado en una versión modificada del kernel de Linux 2.6 el cual utiliza para controlar servicios del núcleo del sistema como pueden ser la seguridad, gestión de memoria o gestión de procesos. A partir de la versión 4.0 (Ice Cream Sandwich) está basado en la familia 3 del kernel de Linux. Actualmente, se puede encontrar presente en teléfonos móviles, PDAs, Tablets e incluso en Netbooks.

Ilustración 8. Logotipo Android

Es una plataforma de código abierto distribuida bajo la licencia Apache 2.0 por lo que su distribución es libre y posibilita el acceso y modificación de su código fuente. Inicialmente fue desarrollado por Google, para más tarde unirse a la Open Handset Alliance (de la cual, Google también forma parte) que está integrada por T-Mobile, Intel, Samsung, HTC o Nvidia entre otros. Sin embargo, Google ha sido la compañía que ha publicado la mayor parte del código fuente bajo la licencia Apache.

En cuanto al ámbito del desarrollo de software para esta plataforma, a los programadores se les proporciona de manera gratuita el SDK y un plugin que se integra dentro del entorno de desarrollo de Eclipse, donde se incluyen todas las APIs necesarias además de un potente emulador integrado para facilitar las pruebas de la aplicación y que ofrece una gran cantidad de posibilidades.

Sobre el campo de acción que tiene Android, hay que destacar que Google siempre ha proclamado que sus objetivos con la plataforma no es que se convierta simplemente en un sistema operativo, lo que pretenden con Android es reunir todos los elementos necesarios para que los desarrolladores tengan acceso a todas y cada una de las funciones que ofrece un

dispositivo móvil de manera sencilla, es decir, se quiere estandarizar el desarrollo de aplicaciones para dispositivos móviles con las ventajas que ello conlleva.

• 3.1.1 Historia de Android

En este apartado vamos a hacer un pequeño repaso por el breve camino que ha recorrido el sistema operativo hasta el momento. También se detallarán las versiones por las que ha pasado y cuales son algunas de las que están por llegar.

Como ya se ha comentado, en octubre de 2008 se presentaba el teléfono G1 cuya principal característica era su sistema operativo, Android 1.0. A partir de este momento, y aunque la primera versión presentaba algunas deficiencias respecto a su competidor más directo, el iPhone de Apple, la plataforma comenzó a crecer a un ritmo vertiginoso. Gran cantidad de compañías comenzaron a presentar sus nuevos teléfonos con Android que mejoraba en cada nueva versión de la plataforma y que han llevado a convertirla en la preferida por los fabricantes de smartphones.

A continuación se muestra la evolución del sistema operativo.

Android 1.0	23 de septiembre de 2008
Android 1.1	9 de febrero de 2009
Android 1.5 (Cupcake)	30 de abril de 2009
Android 1.6 (Donut)	15 de septiembre de 2009
Android 2.0 – 2.1 (Eclair)	26 de octubre de 2009 – 12 de enero de 2010
Android 2.2 – 2.2.3 (Froyo)	20 de mayo de 2010 – 21 de Noviembre de 2011
Android 2.3 – 2.3.7 (Gingerbread)	6 de diciembre de 2010 – 21 de septiembre de 2011
Android 3.0 – 3.2.6 (Honeycomb)	22 de febrero de 2011 – febrero de 2012
Android 4.0 – 4.0.4 (Ice Cream Sandwich) basado en kernel 3.0.1	19 de octubre de 2011 – 29 de marzo de 2012

Android 4.1 – 4.3 (Jelly Bean)	9 de julio de 2012 – 24 de julio de 2013
Android 4.4 (KitKat)	31 de octubre de 2013

Tabla 4. Evolución del sistema operativo Android

Muchas de estas versiones han convivido durante largos lapsos de tiempo. Tanto es así, que aún muchos fabricantes comercializan dispositivos móviles de gama media preparados para la versión 2.3.7. La tendencia, sin embargo, es a hacer que el hardware soporte paulatinamente las últimas versiones de la plataforma. En la siguiente ilustración se puede ver la presencia a día uno de agosto de 2013 por versión de Android en el Google Play:

Version	Codename	API	Distribution
2.2	Froyo	8	1.1%
2.3.3 - 2.3.7	Gingerbread	10	17.8%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	14.3%
4.1.x	Jelly Bean	16	34.4%
4.2.x		17	18.1%
4.3		18	8.9%
4.4	KitKat	19	5.3%

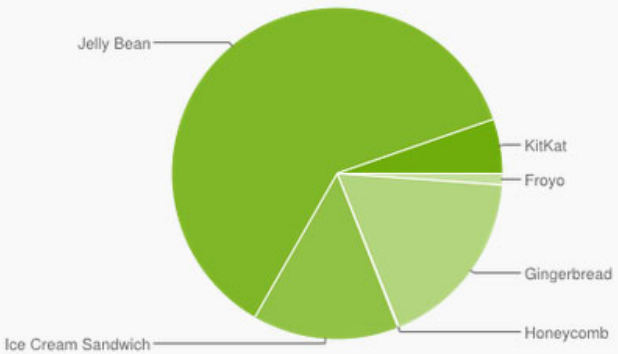


Ilustración 9. Versiones de plataforma en Google Play (Google Inc., 2014)

3.2 Arquitectura de Android 4.x

Android es un sistema diseñado por capas. Como se ha dicho anteriormente, utiliza el kernel de Linux 3.0.x que le da acceso a la parte hardware de los dispositivos a la par que le permite ser compatible con muchos de los drivers creados para Linux. Esta arquitectura está perfectamente plasmada en el diagrama que encontramos en la siguiente figura. Cada sección, será explicada con mayor detalle a continuación.

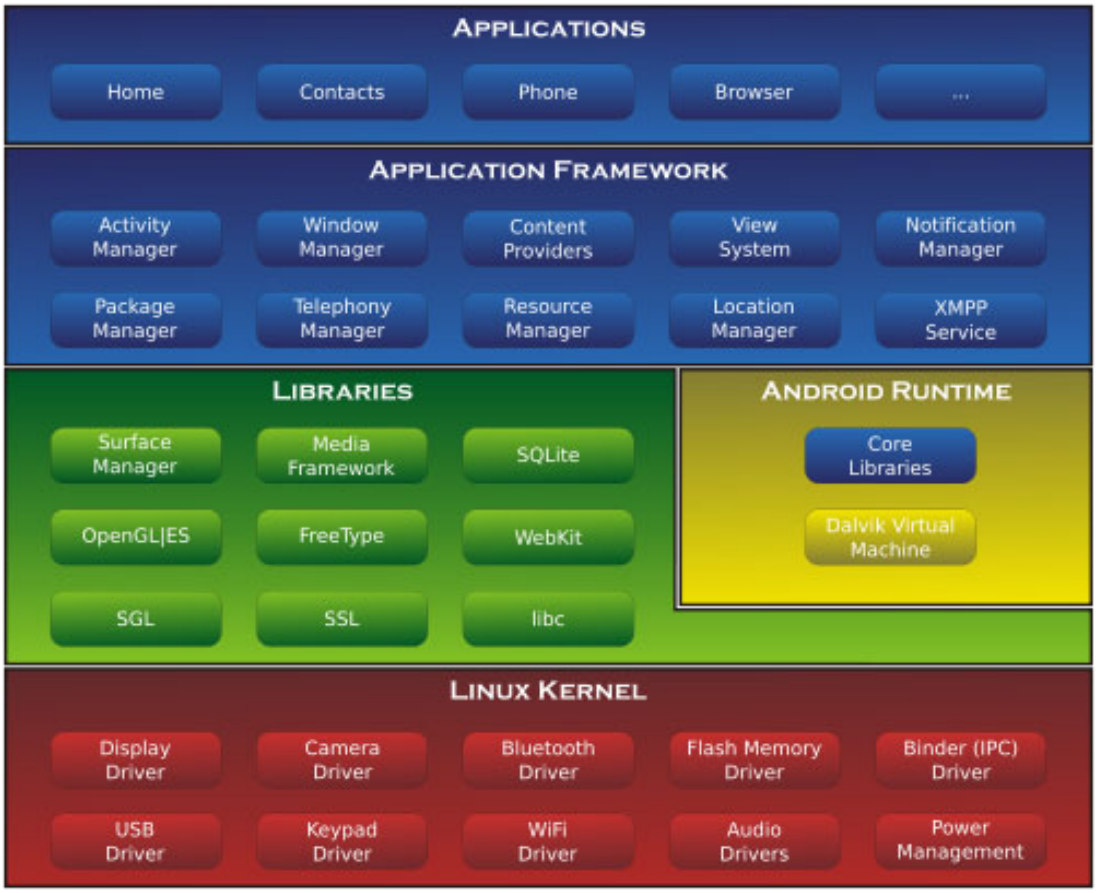


Ilustración 10. Arquitectura de Android 4.3

Aplicaciones

Todas las aplicaciones, tanto las incluidas en el propio Android como las creadas por desarrolladores, están escritas en lenguaje Java y pueden estar compuestas por 5 bloques de los que hablaremos más adelante: *Activity, Intent, Broadcast, Services y Content Providers*, pero no tienen que aparecer todos ellos por obligación, simplemente se utilizarán los necesarios para llevar a cabo los objetivos para los que fue diseñada dicha aplicación. Entre las aplicaciones ya instaladas en el sistema se puede encontrar un cliente de correo, navegador web, gestor de contactos, mapas o calendario entre otras.

Framework

El marco de aplicaciones da acceso completo a los programadores a las mismas APIs utilizadas por las aplicaciones básicas. La arquitectura está diseñada para que la reutilización de componentes sea sencilla, ya que cualquier aplicación puede dar un perfil público a sus datos o capacidades para que otra aplicación haga uso de esas cualidades. Todo esto es posible partiendo de la base de que las normas de seguridad impuestas por el framework no se vean comprometidas, de esta manera se consigue incorporar un importante grado de reutilización de código. En la figura anterior se pueden ver algunas de las librerías más importantes de esta capa.

Bibliotecas

El sistema incluye un conjunto de librerías en C y C++ que proporcionan la mayor parte de las funcionalidades presentes y que son utilizadas por varios de los componentes del sistema Android. Estas capacidades son expuestas a los desarrolladores a través del framework descrito anteriormente para que puedan ser utilizadas. En la siguiente tabla se muestran algunas de las librerías más importantes.

Nombre de la librería	Descripción
System C library	Una implementación de la librería estándar de C (libc) optimizada para su uso en dispositivos móviles
Media Framework	Esta librería soporta la grabación y reproducción de diversos formatos populares tanto de audio como de video además de imágenes. Algunos de estos formatos son: MPEG4, H.264, MP3, AAC, AMR, JPG y PNG
Surface Manager	Gestiona el acceso al sistema de visualización y realiza las composiciones de capas 2D y 3D de diversas aplicaciones.
LibWebCore	Un moderno motor de navegación web que es utilizado por el navegador de Android.
SGL	El motor básico de gráficos en 2D.
3D libraries	Esta implementación está basada en las APIs de OpenGL ES 2.0. La librería utiliza hardware para la aceleración 3D si se encuentra disponible en el dispositivo o un software altamente optimizado en caso de no existir hardware para el 3D.
FreeType	Librería para el proceso de renderización de mapas de bits y de fuentes de texto.
SQLite	Gestor de bases de datos relacionales que se encuentra disponible para todas las aplicaciones.

Tabla 5. Librerías Android

Android runtime

Siguiendo el diagrama de la arquitectura mostrado en la figura 15, al mismo nivel que las librerías de Android, se situa el Android runtime o entorno de ejecución, compuesto por librerías que proporcionan la mayor parte de las funcionalidades de Java además de incluir la máquina virtual. Al ejecutarse, cada aplicación crea su propio proceso con su propia instancia de la máquina virtual Dalvik, que ha sido escrita de manera que un mismo dispositivo pueda lanzar múltiples instancias de dicha máquina virtual de forma eficiente

Kernel de Linux

La capa más cercana al hardware del dispositivo corresponde al núcleo de Android que está basado en el kernel de Linux 3.0.x. Android utiliza este kernel como una abstracción del hardware disponible en cada terminal conteniendo los drivers necesarios para utilizar cualquier parte de este hardware a través de las llamadas correspondientes. Android también depende de este kernel para gestionar los servicios base del sistema como pueden ser la seguridad, gestión de memoria, de procesos, la pila de red y el modelo de driver.

3.3 Dalvik VM

Los requisitos hardware inherentes a los dispositivos móviles obligan a que los recursos sean bastantes limitados. Actualmente los móviles de gamas medio altas tienen capacidades de proceso y memoria RAM que hasta hace poco tiempo eran típicas de ordenadores de sobremesa. Aun así, existen muchos terminales de coste mucho más bajo y recursos mucho más modestos. Por ello Android necesita una máquina virtual con un importante grado de optimización de recursos que se traduzca en una mejora de la usabilidad.

• 3.3.1 Diferencias con la máquina virtual Java

Para conseguir el objetivo antes citado, Google tomó la decisión de apoyarse en el trabajo de Dan Bornstein, quien en colaboración con ingenieros de la compañía escribió el DalvikVM, una máquina virtual que gracias a su diseño consigue reutilizar la información duplicada reduciendo el espacio de memoria utilizado.

Dalvik es un intérprete que solo ejecuta sus propios archivos .dex (Dalvik Executable). Este formato está optimizado para realizar un uso eficiente de la memoria, objetivo que consigue delegando en el kernel la gestión de hilos, de memoria y de procesos. Como se puede observar en la siguiente figura, toma los archivos generados por las clases Java y los combina en uno o varios archivos .dex. De esta manera y sin comprimir ninguna clase, consigue reducir el uso de memoria.

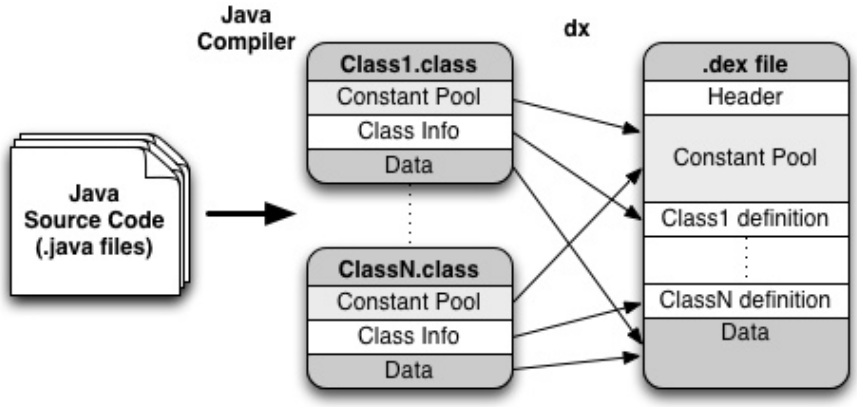


Ilustración 11. Formato de archivos .dex

Además de esta estructura innovadora y dispar con respecto a la máquina virtual Java, Dalvik presenta otro tipo de mejoras que consiguen optimizar aún más el uso de los recursos hardware del sistema. Estas son algunas de esas características:

- Dalvik no es una máquina virtual basada en pila, usa por el contrario una arquitectura basada en registro.
- La máquina virtual ha sido comprimida para usar menos espacio

- Las constantes usan sólo índices de 32 bits para simplificar el intérprete.
- Los Java bytecodes estándar ejecutan instrucciones de pila de 8 bits. Las variables locales han de ser copiadas a la pila de operandos una a una y por instrucciones separadas. Dalvik, en cambio, usa sus propias instrucciones de 16 bit que actúan directamente sobre las variables locales. De esta manera el set de instrucciones de Dalvik es menor y la velocidad del intérprete aumenta.

El diseño de Dalvik permite a un terminal lanzar varias instancias de la máquina virtual de manera eficiente.

Como cada aplicación corre con su propia copia de Dalvik, el recolector de basura no es único. El problema es que, al compartir memoria, la estrategia de recolección ha de estar al tanto de esto. Para evitar problemas se mantienen bits de marca a un determinado objeto que está usado por un proceso. Estos bits de marca se mantienen en una zona de memoria común. Dichos bits simplemente informan al recolector de basura de si ese objeto está en uso por algún proceso o no.

Al haber múltiples instancias de la máquina virtual corriendo en el sistema, hubo que solucionar el problema del tiempo de arranque de la misma. La solución elegida fue el concepto que Android denominó Zygote, que básicamente es una instancia de la máquina virtual que se inicia en tiempo de arranque del sistema y queda a la escucha de comandos. Cuando el sistema necesita una máquina virtual nueva para ejecutar una aplicación, envía un comando a Zygote, y este devuelve un `fork()` de sí mismo.

Por último hay que señalar que el código final ejecutable de Android está constituido por los archivos .dex reunidos en un archivo .apk (Android package).

3.4 Componentes

En este apartado hablaremos sobre los componentes o bloques que se pueden encontrar en cualquier aplicación de Android. No es obligatorio implementarlos todos, por lo que podríamos decir que las aplicaciones están compuestas por una combinación libre de los bloques o componentes descritos a continuación. Todos estos componentes deben ser declarados de forma explícita en el fichero `AndroidManifest.xml` donde se encuentran definidos otros datos importantes como los permisos, siendo un fichero básico en cualquier aplicación. Son `Activity`, `Service`, `BroadcastReceiver` y `ContentProvider`.

Existen otros componentes de segundo nivel, pero mencionaremos aquí al `Fragment`, que sin ser un componente principal tiene ciclo de vida propio y responde a sus propios eventos. Lo explicaremos más adelante.

• 3.4.1 Activity

Las actividades representan el componente principal de la interfaz gráfica de una aplicación Android. Se puede pensar en una actividad como el elemento análogo a una *ventana* en cualquier otro lenguaje visual. Una actividad refleja una determinada actividad llevada a cabo por una aplicación, lleva asociada la interfaz de usuario representada por la clase View y sus derivados. Se implementa mediante la clase del mismo nombre `Activity`.

La mayoría de aplicaciones están compuestas de varias pantallas con diferentes objetivos, por lo tanto, estarán compuestas de varias actividades. El paso de una pantalla a otra se consigue mediante la inicialización de una nueva actividad, cuando una nueva ventana se abre, la actividad anterior queda en pausa dentro de la pila pudiendo el usuario navegar entre actividades previamente abiertas.

El concepto de *Intent* se encuentra estrechamente ligado con la inicialización de las actividades. El concepto de intent se puede definir como la intención de realizar una acción (en este caso, comenzar una nueva actividad). Al lanzar un intent, una aplicación puede llamar a otra aplicación distinta que sea capaz de realizar la acción solicitada, como por ejemplo escribir un mensaje de texto, realizar una llamada o abrir una URL en el navegador web.

• 3.4.2 Service

Los *servicios* son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios o demonios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar notificaciones no intrusivas, o incluso mostrar elementos visuales si se necesita en algún momento la interacción con del usuario para obtener una confirmación.

El ejemplo típico que se da para comprender el concepto de servicio es el reproductor de música que viene incluido en el sistema operativo. Esta aplicación está dotada de una interfaz gráfica (o Activity) donde se permite al usuario elegir entre una lista de canciones, generar listas de reproducción así como el resto de acciones típicas de estos reproductores. Sin embargo, en el momento en que el usuario comienza a reproducir un archivo de audio, puede seguir navegando entre las canciones e incluso comenzar a utilizar otras aplicaciones. Esto se debe a que la reproducción se realizar mediante un servicio y se ejecuta en background. De igual manera que las actividades, se implementa mediante la clase con su mismo nombre.

• 3.4.3 Broadcast Receiver

Un broadcast receiver es un componente que responde a mensajes o eventos globales generados por el sistema. -Algunos ejemplos de mensajes del sistema a los que puede responder un broadcast receiver:

- La pantalla se ha apagado/encendido
- La batería está baja
- Aviso de llamada entrante

- Aviso de SMS entrante
- …

Las aplicaciones pueden también iniciar mensajes para que sean capturados por broadcast receiversque estén activos y esperando ese intent en particular.

Este componente tampoco tiene una interfaz gráfica asociada, al igual que el Service. Al igual que los componentes anteriores, los broadcast receivers se implementan a través de la clase con su mismo nombre.

• 3.4.4 Content provider

Un *content provider* es el mecanismo que se ha definido en Android para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de la aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, la aplicación podrá acceder a los datos de otra a través de los *content provider* que se hayan definido.

Existe una serie de content providers ya definidos que se encuentran implementados y que permiten compartir todo tipo de datos, información de los contactos, imágenes, video, mensajes de texto o incluso audio. Estos se encuentran dentro del paquete `android.provider` ofreciendo además la posibilidad de crear content providers propios.

Los content provider son objetos de la clase `ContentProvider`, que se encuentra localizada en el paquete `android.content`. Cada objeto de esta clase lleva asociada una URI única que sirve para identificarlo y a través de la cual el resto de aplicaciones deben acceder a él. Cuando creamos un nuevo content provider en la aplicación, los datos que vayamos a hacer públicos se almacenarán habitualmente como una tabla de una base de datos SQLite donde cada columna se refiere a un tipo de dato y cada fila representa un registro.

Suponiendo que el content provider definido por el sistema para los contactos tuviera el formato Número, Etiqueta y Nombre la tabla resultado de utilizar dicho proveedor de contenido sería la siguiente. La columna de identificación es impuesta por la propia base de datos como vemos en la tabla.

_ID	app_id	frequency	Locale	word
1	User1	100	en_US	mapreduce
2	User14	200	es_ES	precompiler
3	User2	225	en_UK	applet

Tabla 6. Ejemplo de tabla de content provider

En el siguiente ejemplo, se muestra la manera de acceder a dicha información utilizando la URI que identifica al content provider definido anteriormente. Se realiza la consulta mediante el método `managedQuery`, aunque existen otros métodos para este fin.

```
Cursor mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // La URI del ContentProvider
    mProjection,                        // Columnas que devolver
    mSelectionClause                    // Criterio de selección
    mSelectionArgs,                     // Argumentos de selección
    mSortOrder);                        // Ordenación
```

Fragmento de código 1. ContentProvider

El cursor así obtenido puede ser recorrido para ir recogiendo cada uno de los registros de la tabla. Esta llamada al Content Provider, es equivalente a la siguiente sentencia SQL

```
SELECT _ID, word, locale FROM words WHERE word = <userinput> ORDER BY word
ASC;
```

Fragmento de código 2. Sentencia SQL ejemplo

• 3.4.5 Fragment

Un Fragment es un componente que representa el comportamiento de una porción de interfaz de usuario dentro de un `Activity`. En una misma `Activity` pueden convivir numerosos `Fragments`, formando una interfaz de usuario multi-panel.

Se podría decir que un `Fragment` es una sección modular de una `Activity`, que tiene su propio ciclo de vida, recibe sus propios eventos de entrada y puede ser añadido o quitado del `Activity` padre mientras este aún está ejecutándose. Un `Fragment` puede fácilmente ser reutilizado en otra `Activity` diferente.

Un Fragment siempre ha de estar embebido en un Activity, y su ciclo de vida es afectado directamente por el de su Activity padre. Si ese Activity pasa al estado onPause, todos los Fragments contenidos en él, pasaran a ese estado también, por ejemplo.

El componente Fragment fue introducido en Android 3.0, en respuesta al problema de crear interfaces de usuario que fueran más dinámicas y flexibles en pantallas grandes, como las tabletas. Lo que ofrece el componente Fragment es facilidad a la hora de crear vistas complejas, son pequeños elementos autónomos dentro de un componente contenedor. Es un ejemplo claro de "Divide y vencerás".

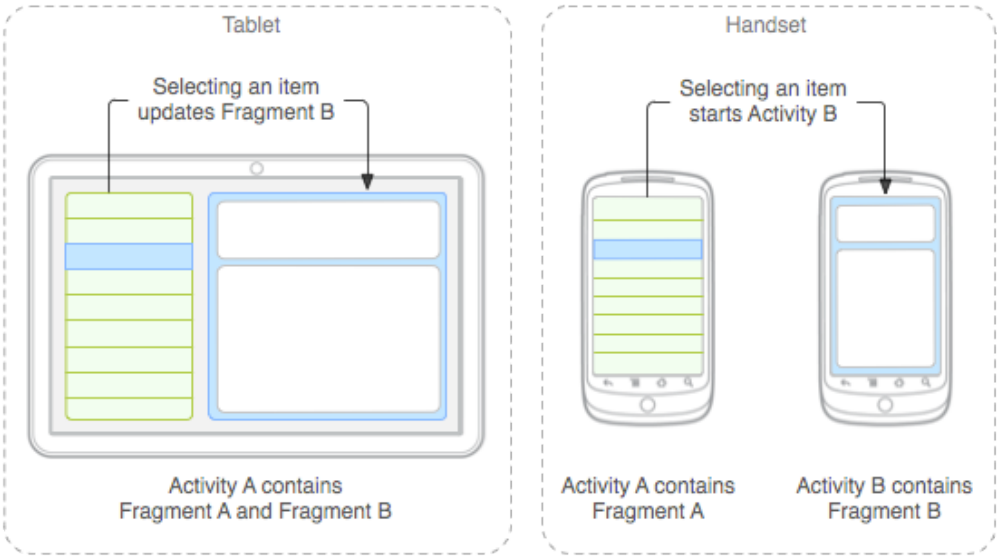


Ilustración 12. Fragmentos en diferentes dispositivos

3.5 Ciclo de vida de una aplicación

Una aplicación Android se ejecuta en su propio proceso Linux. Este proceso es creado cuando parte del código necesita ser ejecutado, y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante y poco usual de Android es que el tiempo de vida de un proceso no es controlado directamente por la aplicación, sino que es el sistema quien determina el tiempo de vida basado en el conocimiento que tiene de las partes de la aplicación, la importancia de la misma y cuánta memoria hay disponible. Por lo tanto, el usuario no tiene conocimiento sobre estas áreas ya que según la naturaleza de Android son tareas del sistema operativo.

Cada componente tiene un ciclo de vida bien definido, a continuación se muestran los ciclos de vida asociados a tres de los componentes más importantes, las actividades, los fragmentos y los servicios. A la hora de crear una aplicación, se debe tener muy en cuenta estos ciclos de vida para que el comportamiento no sea errático.

• 3.5.1 Ciclo de vida del componente Activity

Android maneja las actividades como una pila, cuando una nueva actividad es creada, se coloca en lo más alto de la pila y relega a la actividad anterior a permanecer justo debajo de ella en la propia pila. En la siguiente tabla se muestran en detalle los posibles estados de una actividad junto con su descripción. Todos estos estados podrán ser observados en la figura de la página siguiente.

Estado	Descripción
onCreate	Se ejecuta cuando se crea la Activity por primera vez. Aquí es donde se deberían crear views, linkar datos a listas, en definitiva el proceso de inicialización de la aplicación.
onRestart	Se ejecuta cuando la aplicación se ha cerrado y se va a ejecutar nuevamente.
onStart	Se ejecuta cuando la aplicación aparece visible para el usuario.
onResume	Se ejecuta cuando la Activity interactúa con el usuario. En éste punto la Activity está en la cima de la pila.
onPause	Se ejecuta cuando el sistema está a punto de continuar una Activity anterior. Se utiliza para guardar datos que no se han grabado anteriormente o detener acciones que consuman CPU.
onStop	Se ejecuta cuando la Activity deja de ser visible al usuario. Puede ocurrir porque una nueva Activity ha sido creada, una Activity ya creada pasa a primer plano o ésta está siendo destruida.
onDestroy	Última llamada antes de destruir la actividad. Puede ocurrir porque la actividad está acabando o porque el sistema destruirá la instancia por necesidad.

Tabla 7. Métodos de ciclo de vida de un Activity

Tras la anterior descripción de todos los posibles estados, solo nos faltaría por conocer la relación que existe entre los propios estados para así, a la hora de implementar la aplicación, tomar las decisiones adecuadas respecto a cada situación. El diagrama siguiente nos muestra esta relación.

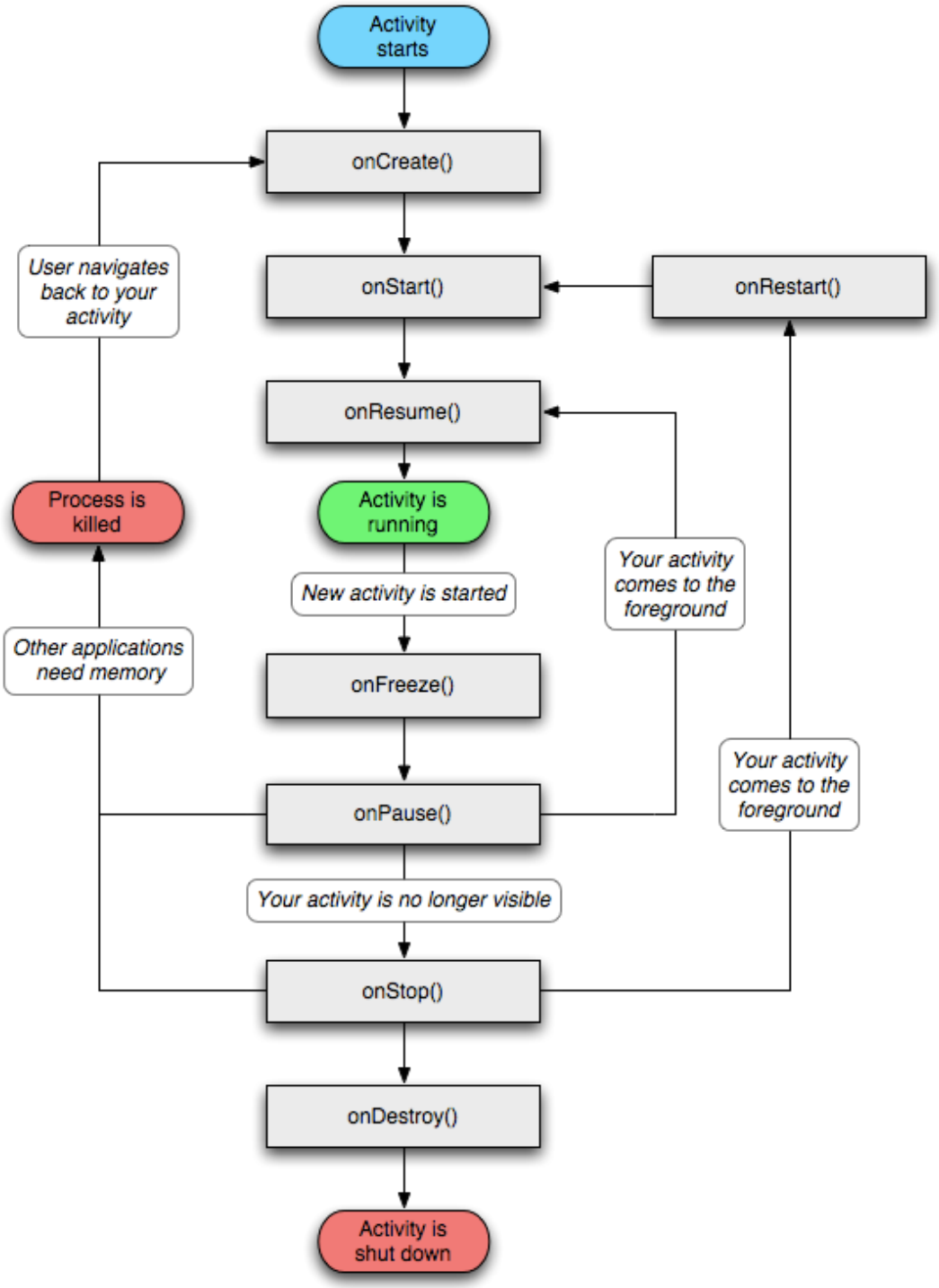


Ilustración 13. Ciclo de vida de una actividad (Google Inc., 2013)

- 3.5.2 Ciclo de vida del componente Fragment

Al igual que en el caso del componente Activity, conviene explicar el ciclo de vida de un Fragment. En la siguiente tabla se pueden ver los estados principales por los que pasa un Fragment.

Estado	Descripción
onCreate	Se ejecuta cuando se crea el Fragment por primera vez. Aquí es donde se deberían crear componentes que se quieran retener durante el ciclo de vida completo.
onCreateView	Se ejecuta cuando el sistema quiere que el fragment dibuje su estructura.
onPause	Se ejecuta cuando el usuario abandona el fragment. No significa que vaya a ser destruido.
onDestroyView	La vista está a punto de ser destruida

Tabla 8. Métodos de ciclo de vida de un Fragment

Para entender mejor el funcionamiento del Fragment, la ilustración de la página siguiente representa las interacciones entre estados del ciclo de vida.

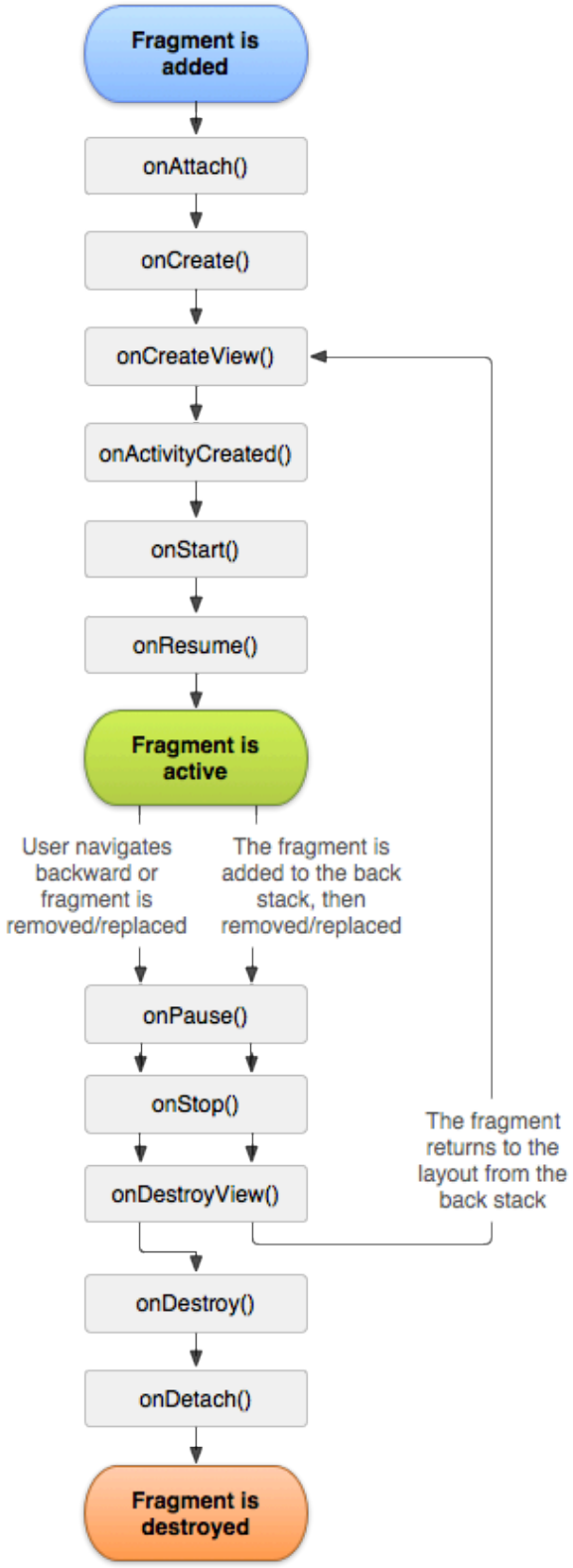


Ilustración 14. Ciclo de vida de un Fragment (Google Inc., 2013)

• 3.5.3 Ciclo de vida del componente Service

De igual manera que una actividad, los servicios tienen un ciclo de vida muy definido por el sistema. Existen ciertos métodos que se pueden implementar para lograr un mayor control sobre este ciclo de vida realizando las acciones oportunas en cada momento del ciclo. Sin embargo, debemos distinguir entre dos formas distintas de iniciar un servicio. A continuación se mostrarán los detalles de ambas.

Servicio creado vía startService

En este caso, el ciclo de vida del servicio ocurre entre la llamada a la función onCreate() y el momento en que onDestroy() devuelve su valor. En la figura se puede observar dicho ciclo de vida.

Servicio creado vía bindService

De esta manera, el cliente establece una conexión con el servicio que es utilizada para llamar al propio servicio. Este formato permite que varios clientes se conecten al mismo servicio, siempre que este lo permita.

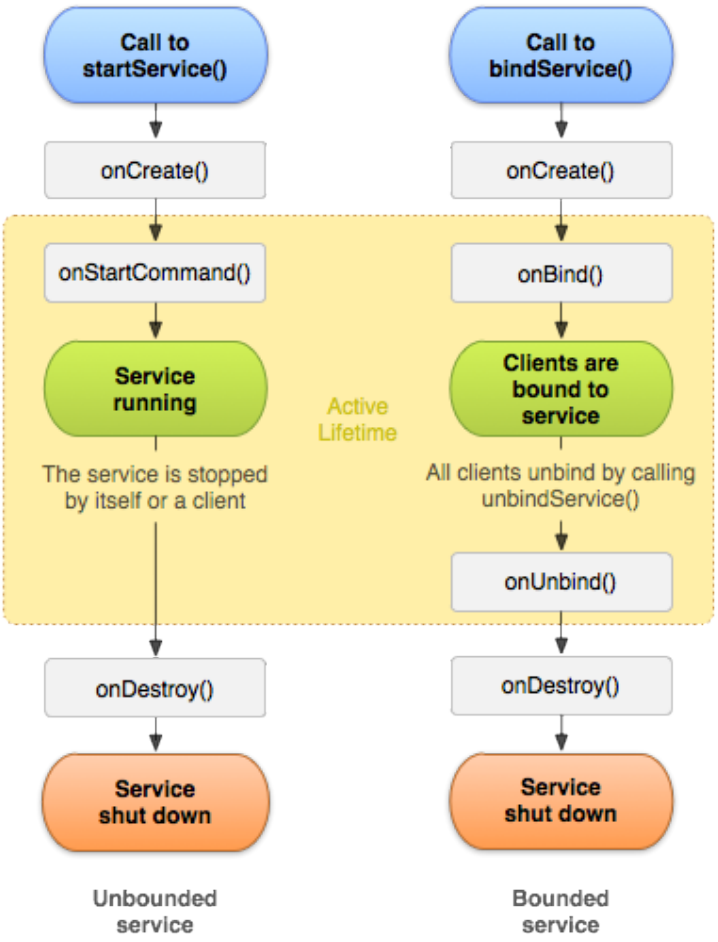


Ilustración 15. Diferentes ciclos de vida de un Service (Google Inc., 2013)

3.6 Seguridad

La mayor parte de las medidas de seguridad que se puede encontrar en Android provienen del kernel de Linux, que como ya se dijo, constituye el núcleo del sistema operativo Android. Cada aplicación se ejecuta en su propio proceso, y es dicho proceso el que nos ofrece un entorno seguro de ejecución, ya que a priori, no se permite realizar ninguna operación que pueda causar un efecto negativo en otras aplicaciones o en el propio terminal.

Para las aplicaciones está prohibido por el sistema ejercer acciones restringidas como realizar llamadas de teléfono, leer y escribir la información de contactos, tener acceso a internet o utilizar un elemento hardware del dispositivo. La única manera de tener acceso a estas funcionalidades es declarándolas explícitamente durante el desarrollo de la aplicación para que el sistema nos autorice a ello. Este es un ejemplo de configuración para que el sistema nos permita acceder a internet y hacer llamadas utilizando el teléfono:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Fragmento de código 3. Declaración de permisos

Además, Android requiere que las aplicaciones sean firmadas digitalmente para ser publicadas en el Google Play, esto se consigue mediante un certificado digital que contiene todos los datos relativos al creador de dicha aplicación. Uno de los principales beneficios que se desprenden de esta modalidad, es que una aplicación no podrá ser actualizada si dicha versión no fue publicada por el autor original.

Siguiendo con la mecánica de que cada aplicación se ejecuta en su propio proceso encontramos otra característica importante de seguridad, y es que debido a esta ejecución una aplicación no puede acceder a los datos de otra (a menos que se declaren Content providers como se vio anteriormente). Este aislamiento de procesos se consigue gracias al sistema operativo, que limita el radio de acción de las aplicaciones siempre que no existan solicitudes explicitas de intenciones. Es decir, se proporcionan los métodos necesarios para realizar peticiones y compartir información, pero siempre de forma controlada y bajo la supervisión de Android.

3.7 Almacenamiento de la información

Android provee de distintas opciones a la hora de almacenar datos de una aplicación de forma persistente en el sistema. Dentro de esta variedad de opciones, es importante que se elija la que mejor se acomoda a las necesidades teniendo que valorar si los datos a guardar serán de dominio público (accesibles por el resto de aplicaciones y por el usuario) o si por el contrario el perfil es privado y solo la aplicación tendrá acceso a los mismos. A continuación se presentan las opciones más comunes junto con una descripción de cada una.

• 3.7.1 Shared Preferences

El sistema operativo ofrece la clase `SharedPreferences` que se podrá utilizar en la tarea de almacenar las preferencias de usuario en la aplicación, permitiendo guardar y recuperar valores de tipos de datos primitivos como pueden ser booleanos, numéricos de cualquier tipo y cadenas de texto. En Android 4.0 se extendió la cantidad de tipos de datos para incluir estructuras de más alto nivel, como las colecciones. Estos datos serán persistentes aunque la aplicación sea eliminada de la memoria de trabajo.

Para obtener un objeto de la clase `SharedPreferences` en una aplicación se deberá utilizar alguno de estos dos métodos:

- `getSharedPreferences()`. Cuando se necesita utilizar múltiples ficheros de preferencias identificados por el nombre, que será especificado en el primer parámetro.
- `getPreferences()`. Se utilizará este método si solo se va a utilizar un fichero de preferencias. En este caso, no es necesario darle nombre.

A nivel interno funciona como un HashMap, en el que almacenan las preferencias del modo <clave, valor>.

• 3.7.2 Almacenamiento interno

Al desarrollar una aplicación se tiene la posibilidad de guardar algunos datos en la memoria interna del teléfono. Por defecto, los datos almacenados en esta memoria serán de carácter privado, siendo la aplicación la única con permisos sobre ellos y evitando así que tanto otras aplicaciones, como el usuario, puedan acceder a los mismos. Se puede, sin embargo, escribir en esta zona de almacenamiento con permisos de lectura y/o escritura para el resto de aplicaciones.

Recordemos que el sistema operativo Android se basa en Linux, por lo que realmente lo que hacemos con esto es cambiar los permisos de lectura y escritura para el propietario, grupo y resto.

• 3.7.3 Almacenamiento externo

Todos los dispositivos Android soportan el uso de un almacenamiento externo compartido, pudiendo ser un medio desmontable como una tarjeta SD, o una parte interna de la memoria dedicada para esta tarea. En cualquier caso, toda información almacenada de forma externa, será accesible y podrá ser modificada por el usuario así como por otras aplicaciones. Es importante tener en cuenta que estos medios externos, pueden ser extraídos por el usuario del dispositivo Android, por lo que no es conveniente almacenar datos necesarios para la ejecución de la aplicación

Por este motivo, antes de utilizar el almacenamiento externo, se debe comprobar la disponibilidad para lectura y escritura del mismo.

• 3.7.4 Almacenamiento con base de datos SQLite

El sistema operativo Android ofrece soporte total para el uso de bases de datos SQLite en las aplicaciones. Este modelo es un sistema de gestión de bases de datos contenido en una pequeña librería en C y que es compatible con ACID. Permite la creación de bases de datos relacionales, navegación entre tablas, ejecución de sentencias SQL además del resto de funcionalidades propias del sistema SQLite.

Esta base de datos se encuentra a disponibilidad del usuario y según la documentación, la forma recomendada para crear una nueva base de datos SQLite es crear una subclase del tipo `SQLiteOpenHelper` y sobrescribir el método `onCreate`, en el cual se podrá ejecutar comandos SQL para la creación de tablas.

A continuación se muestra un ejemplo sobre cómo generar esta subclase y crear una nueva base de datos junto con una tabla.

```
public class BDHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_NAME = "dictionary";
    private static final String KEY_WORD = "word";
    private static final String KEY_DEFINITION = "definition";
    private static final String TABLE_CREATE =
                    "CREATE TABLE " + TABLE_NAME + " (" +
                    KEY_WORD + " TEXT, " +
                    KEY_DEFINITION + " TEXT);";

    public BDHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

3.8 Organización de un proyecto de Android

En este apartado se explicará la estructura de un proyecto de Android en el entorno de desarrollo integrado (IDE) Eclipse.

Al crear un nuevo proyecto Android, en el workspace de Eclipse se genera un árbol de directorios cuyo primer nodo es una carpeta con el nombre del proyecto que contiene una serie de subdirectorios y archivos que constituyen el esqueleto básico del proyecto. A continuación se puede observar dicha estructura y tras la figura se explicará con mayor detalle el contenido y propósito de cada directorio.

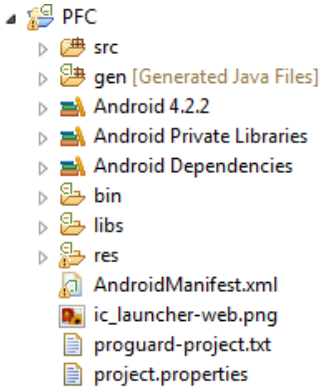


Ilustración 16. Árbol de directorios de un proyecto

Carpeta src

En este directorio es donde se encuentran los paquetes por los que está formada la aplicación. Dentro de cada paquete se pueden encontrar los archivos .java que hemos implementado.

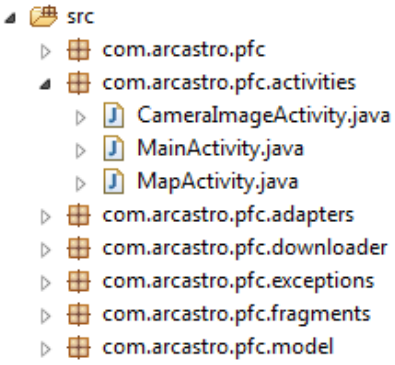


Ilustración 17. Carpeta src

Carpeta gen

Dentro de la carpeta gen se encuentran archivos autogenerados por Eclipse. El más importante de estos archivos es el archivo R.java que no debe ser modificado bajo ningún concepto. En él se establece la relación entre los recursos, tales como archivos de imagen, cadenas y layouts mediante un identificador único en la aplicación, con el recurso físico en sí.

El archivo R.java puede no ser único. Si importamos una librería con sus propios recuros, tendremos otro paquete en la carpeta gen bajo el cual habrá otro archivo R.java. A la hora de referenciar un recurso, si la clase no está en el mismo paquete que el R.java, habrá que importarlo como si de una clase normal se tratara.

En la siguiente captura, puede verse que este es el caso que acabo de mencionar. Hay un R.java en el paquete `com.arcastro.pfc` y otro en el `com.google.android.gms`, que es la librería importada de mapas de Google.

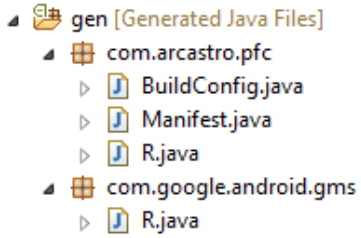


Ilustración 18. Carpeta gen

Android 4.2.2

Aquí puede verse la versión de Android con la que estamos trabajando y el jar utilizado

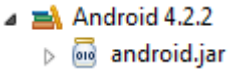


Ilustración 19. Carpeta Android

Android Private Libraries y Android Dependencies

En estas carpetas simbólicas se pueden encontrar librerías externas que usa el proyecto, y en el caso de Android Dependencies, otros proyectos de Android en los que el proyecto se apoya.

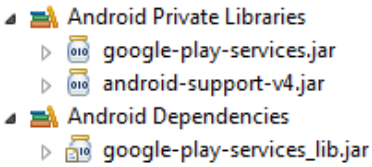


Ilustración 20. Carpetas Private libraries y Dependencies

Carpeta bin

Esta carpeta contiene todos los archivos binarios generados por el Eclipse después de compilar y linkar todas las clases y librerías. Entre ellos puede verse el archivo classes.dex, que como expliqué es el archivo compilado para la máquina virtual Dalvik y el archivo PFC.apk, que es el paquete de aplicación completo.

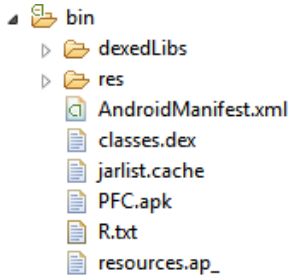


Ilustración 21. Carpeta bin

Carpeta libs

En esta carpeta se puede hacer copiar y pegar directamente archivos .jar que queramos añadir al proyecto. De esta manera Eclipse automáticamente lo añade al classpath para compilar.

Carpeta res

En esta carpeta es donde están todos los recursos de la aplicación.

- En las carpetas drawable se encuentran los archivos de imagen o archivos xml que hacen referencia a un objeto visual, como un gradiente o un selector.
- La subcarpeta layout contiene todas las vistas declaradas en XML de la aplicación.
- La subcarpeta menú contiene las estructuras de menú contextuales de cada Activity, si hubiera alguna.
- Las carpetas values contienen los valores ya sean de dimensiones, márgenes, atributos, cadenas de caracteres etc.

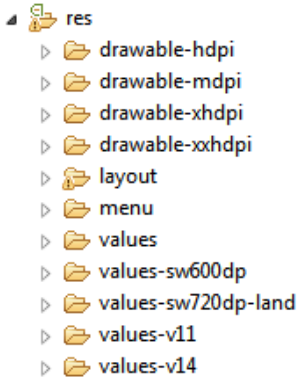


Ilustración 22. Carpeta res

AndroidManifest.xml

El archivo AndroidManifest merece una mención aparte y se explicará más extensamente en el punto 3.10.

3.9 Interfaz de usuario

La interfaz de usuario de una aplicación Android, se define, como ya se ha comentado, en los archivos XML incluidos en el directorio *res/layout*. Cada pantalla de la aplicación, tendrá que ser definida en un archivo distinto.

Debido a que el diseño de la interfaz gráfica mediante código podría resultar complicado a la par que ineficiente, Android incluye los archivos XML para este propósito. El sistema tiene definidos una gran cantidad de elementos diseñados que son subclases de la clase genérica *View*, como por ejemplo Button, TextView, ImageView, EditText, etc. Además existen los *Viewgroups* que son conjuntos de vistas. Un ejemplo claro sería el widget ListView.

Los layouts son los encargados de contener a las vistas que acabamos de explicar. Dentro de un *layout* se incorporan todos los elementos (o *views*) que conformarán la pantalla. Existen varios tipos de layouts para ordenar dichos elementos y que pasamos a detallar.

Linear Layout

Apila sus elementos de forma vertical u horizontal. Estos hijos pueden ser views, viewgroups u otros layouts. A continuación tenemos una captura de linear layout.

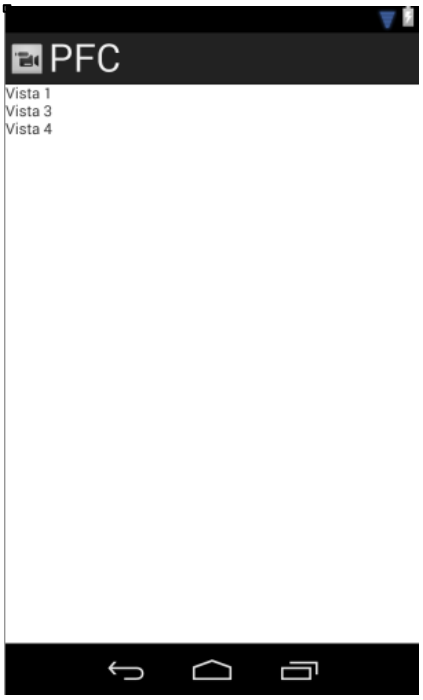


Ilustración 23. LinearLayout

Y aquí se puede ver el código en XML de esta composición:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista1" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista2" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista3" />

</LinearLayout>
```

Fragmento de código 4. LinearLayout XML

Relative Layout

En este caso, los elementos se disponen en una posición relativa entre ellos o entre el propio layout. Aquí se puede jugar con la colocación poniendo elementos en cualquier posición alrededor de otros. En la imagen puede verse como el elemento tres se coloca a la derecha del elemento uno aunque se encuentre definido el último.

Ilustración 24. RelativeLayout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista1" />

    <TextView
        android:id="@+id/textView2"
        android:layout_below="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista2" />

    <TextView
        android:id="@+id/textView3"
        android:layout_toRightOf="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista3" />

</ RelativeLayout >
```

Fragmento de código 5. RelativeLayout XML

FrameLayout

En el caso de este ViewGroup, los elementos añadidos a su estructura se van colocando sobre el anterior, en orden de inserción. Es útil para hacer composiciones superponiendo vistas. En la siguiente captura se puede apreciar como las tres vistas están unas sobre las otras.

Ilustración 25. FrameLayout

En el código siguiente vemos el anterior layout declarado:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista1" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista2" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/vista3" />

</FrameLayout>
```

Fragmento de código 6. FrameLayout XML

Una vez se ha definido el layout que se va a utilizar como base, se deben comenzar a definir todos los elementos que incluiremos en dicho layout. El propósito de estos controles no es otro que el de establecer un flujo de información entre el usuario y la aplicación. Para ello existen multitud de controles que ayudan al programador a crear ese flujo de información. A continuación se listarán algunos de los más importantes.

ListView

Esta vista se representa en pantalla como una lista con desplazamiento vertical que es capaz de mostrar los elementos del `Adapter` que tenga asociado. El formato con el que se muestran los elementos puede ser modificado pudiendo incluir cualquier componente gráfico que extienda de la clase `View` dentro de cada uno de ellos.

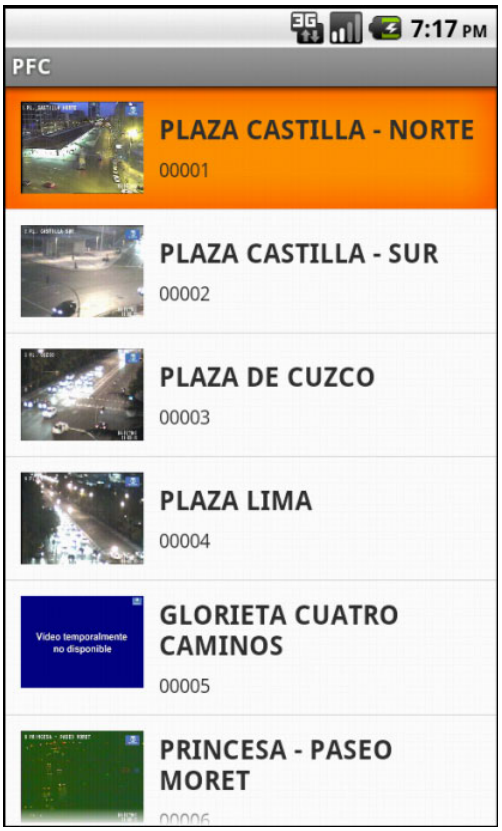


Ilustración 26. ListView

Con su código:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/list_view_cameras"
        android:layout_width="match_parent"
       android:layout_height="match_parent"/>

</RelativeLayout>
```

Fragmento de código 7. ListView XML

Y este es el código de cada uno de los elementos:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingBottom="10dp"
    android:paddingTop="10dp" >

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="10dp"
        android:maxHeight="40dp"
        android:src="@drawable/ic_launcher" />

    <LinearLayout
        android:id="@+id/LinearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_marginLeft="10dp"
        android:layout_marginRight="10dp"
        android:layout_toRightOf="@+id/imageView1"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/titulo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ellipsize="end"
            android:maxLines="2"
            android:textColor="#333333"
            android:textSize="17sp"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/subtitulo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="6dp"
            android:ellipsize="end"
            android:maxLines="1"
            android:textColor="#333333"
            android:textSize="12.5sp" />
    </LinearLayout>

</RelativeLayout>
```

Fragmento de código 8. Item de ListView

Como se puede apreciar, el layout de cada ítem puede ser bastante más complejo que su propia vista padre.

GridView

Esta vista muestra los elementos en una matriz multimensional, de igual manera que las listas, los elementos a mostrar son los incluidos en el `Adapter` asociado con esta vista.

Ilustración 27. GridView

AlertDialog

Es un tipo de ventana que muestra información al usuario, se utilizan para fines muy concretos y existen varios tipos, aunque el más habitual es utilizado con dos botones. En cuanto a los ProgressDialog son ventanas que muestran una animación de progreso de un trabajo que se está realizando en background.

Ilustración 28. AlertDialog

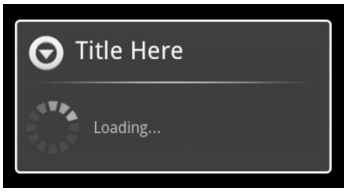


Ilustración 29. ProgressDialog

3.10 Android Manifest

El archivo *AndroidManifest.xml* es absolutamente imprescindible en cualquier aplicación Android. Este archivo generado de forma automática al crear un nuevo proyecto está escrito en XML y describe los componentes de los que consta la aplicación. Esta descripción contiene aspectos como las clases que los implementan, sus requisitos, los datos que pueden manejar o cuando deben ser ejecutados (si al iniciar la aplicación o bajo orden de algún *intent*). En este archivo es donde deben concretarse los permisos que el desarrollador solicita utilizar para su aplicación. Como ya se vio, estos permisos van desde realizar llamadas de teléfono, controlar hardware del dispositivo como la cámara o el vibrador, localización por GPS, acceso a los datos de contactos o acceso a internet.

Aparte de lo mencionado, el manifest hace también lo siguiente:

- Le da nombre al paquete java de la aplicación. El nombre del paquete sirve como identificador único para la aplicación.
- Describe las actividades, servicios, broadcast receivers y content providers de los que se compone la aplicación
- Determina qué procesos contendrán componentes de aplicación.
- Declara los permisos que se requieren para interactuar con los componentes de la aplicación.
- Declara el nivel mínimo de Android API que la aplicación requiere.
- Lista las librerías que deben estar presentes en el proceso de linkado.

En la página siguiente se puede ver un ejemplo real de un archivo AndroidManifest.xml, en concreto el de la aplicación desarrollada para este proyecto.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.arcastro.pfc"
    android:versionCode="1"
    android:versionName="1.0" >

    <permission
        android:name="com.arcastro.pfc.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />

    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="com.arcastro.pfc.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <supports-screens android:compatibleWidthLimitDp="320" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

    <application
        android:name="PFCApp"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyCN2bNAv0YboCLwcyHif9mPnmqUnKYyA8o" />

        <activity
            android:name="com.arcastro.pfc.activities.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="com.arcastro.pfc.activities.CameraImageActivity" />
        <activity android:name="com.arcastro.pfc.activities.MapActivity" />
    </application>

</manifest>
```

Fragmento de código 9. Android Manifest

CAPITULO 4. LECCIONES PROPUESTAS PARA LABORATORIO ANDROID

En este capítulo se expondrá una serie de lecciones incrementales para la introducción al desarrollo bajo la plataforma Android. Para la realización de algunas prácticas me he apoyado en obras de Reto Meier: (Meier, Professional Android 2 Application Development, 2010) (Meier, Professional Android 4 Application Development, 2012).

4.1 Práctica demostrativa 0. entorno de desarrollo y hola mundo

El objetivo de esta práctica introductoria es que el alumno se familiarice con el entorno de desarrollo, sus funciones principales y el desarrollo de una primera aplicación Android de ejemplo.

El entorno de desarrollo es una modificación de Eclipse realizada por Android llamado ADT (Android Developer Tools). Puede ser descargado desde aquí: http://developer.android.com/sdk/index.html

Esta versión modificada incluye plugins de ayuda como el Android SDK Manager, que sirve para administrar los diferentes SDKs, librerías de compatibilidad, etc.

• 4.1.1 Creación de nuevo emulador Android

Este es el icono en la barra de tareas que abrirá la pantalla del Android SDK Manager:  Al pulsarlo podremos ver la siguiente ventana, en la que podremos ver los SDKs que tenemos instalados así como los paquetes que se pueden instalar.

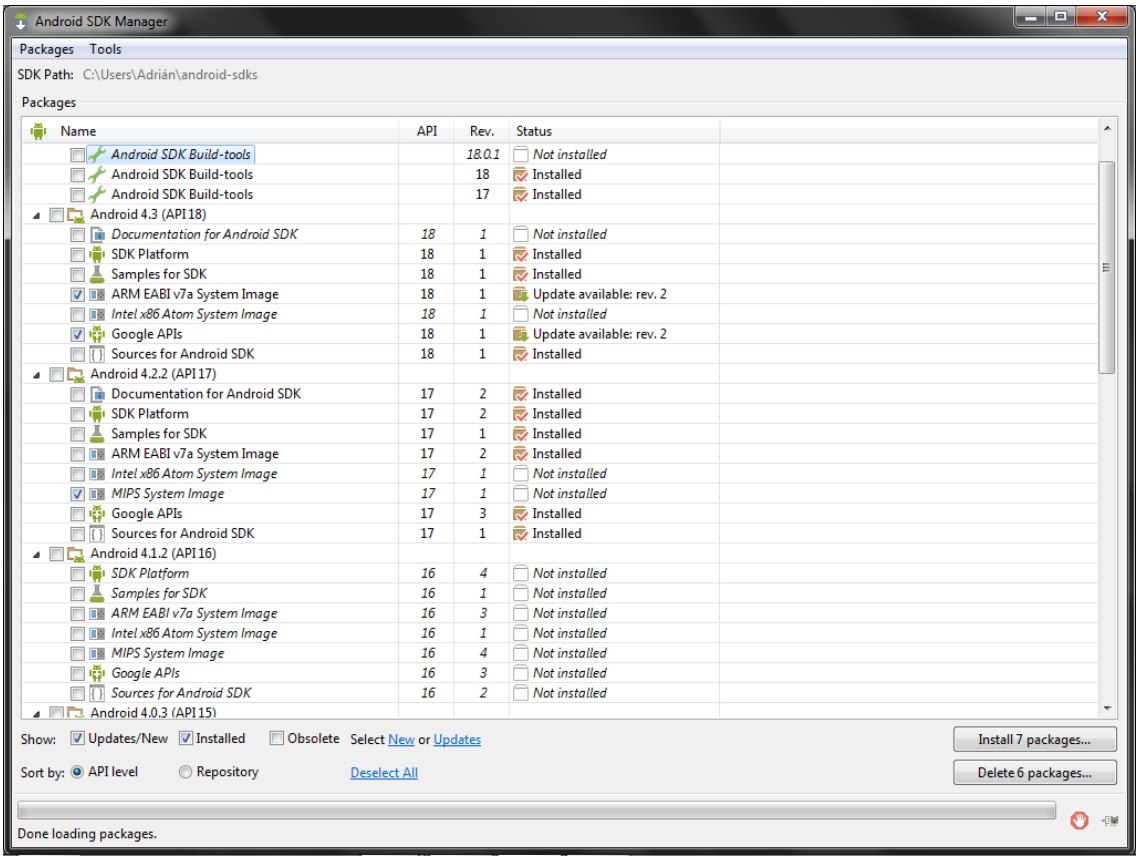


Ilustración 30. Android SDK Manager

Otro plugin interesante que se debe conocer es el Android Virtual Device Manager, que sirve para crear, editar o eliminar los diferentes emuladores que necesitemos para probar la aplicación. Este es su icono:  . Esta es la pantalla:

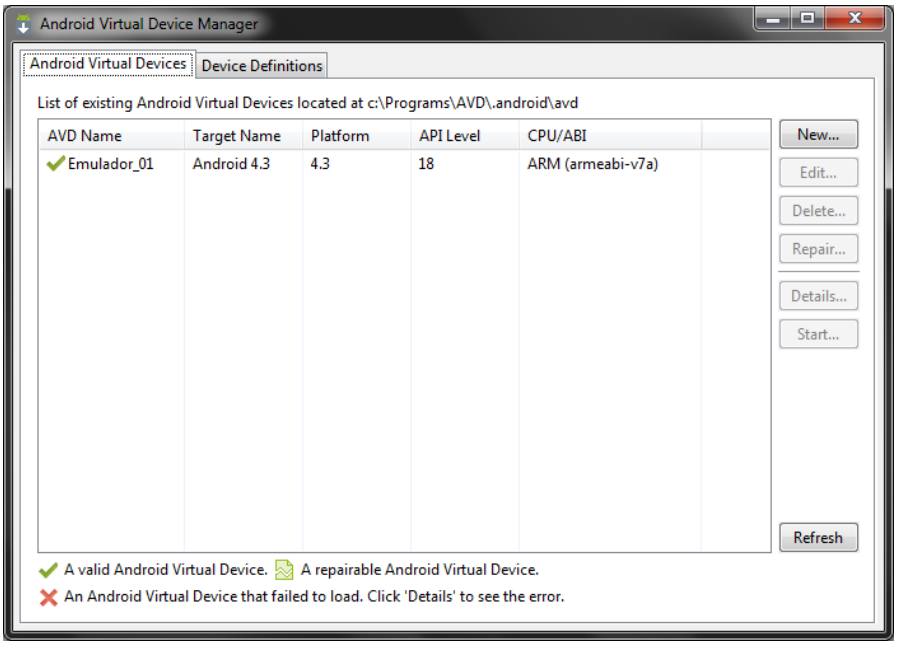


Ilustración 31. Android Virtual Device manager

Para la realización de esta práctica necesitaremos un emulador que se puede crear pulsando en el botón New. La configuración podría ser algo parecido a la siguiente:

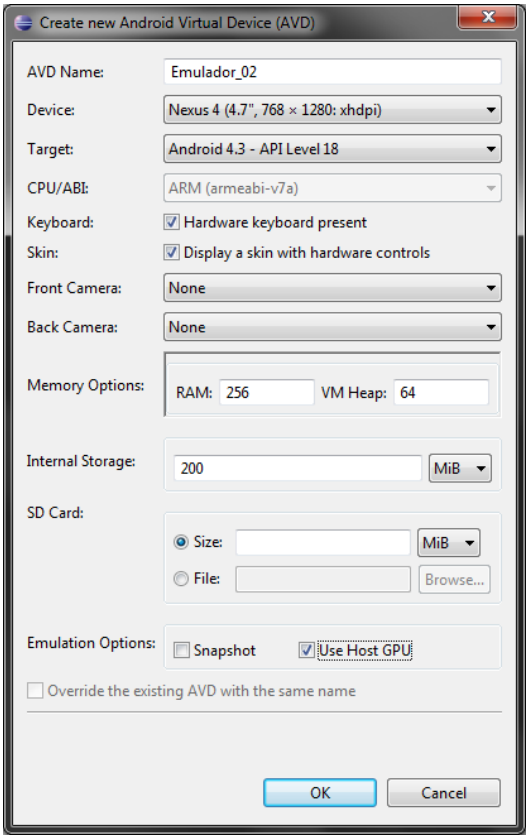


Ilustración 32. Crear nuevo AVD

Una vez creado, si se elige y se pulsa el botón Start, se verá cómo se inicia el emulador completo.

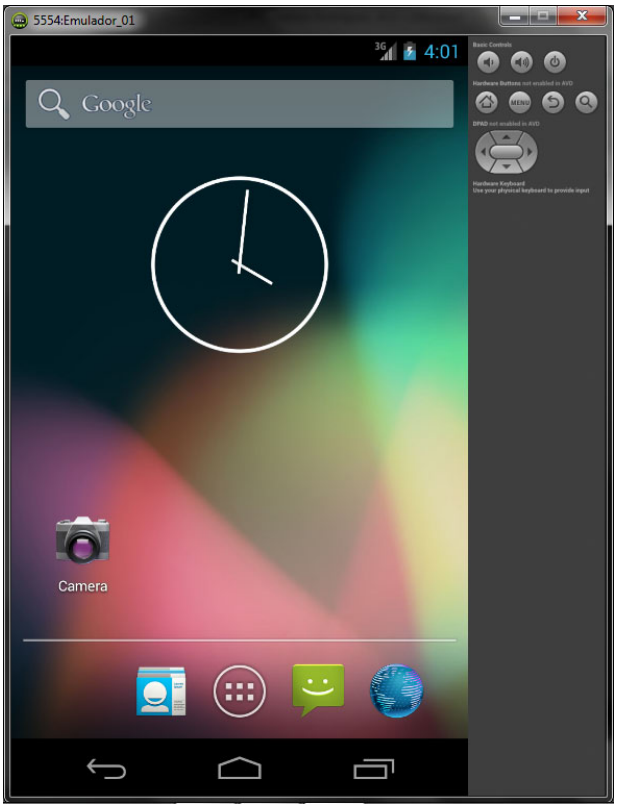


Ilustración 33. Emulador Android corriendo

• 4.1.2 Creación del proyecto

Una vez se tenga el emulador configurado y funcionando, se podrá comenzar con el proyecto. Va a ser una aplicación sencilla con un simple `TextView` que diga "Hola Mundo". Para ello, en el menú File -> New, hay que elegir la opción Android Application Project. Aparecerá la siguiente pantalla:

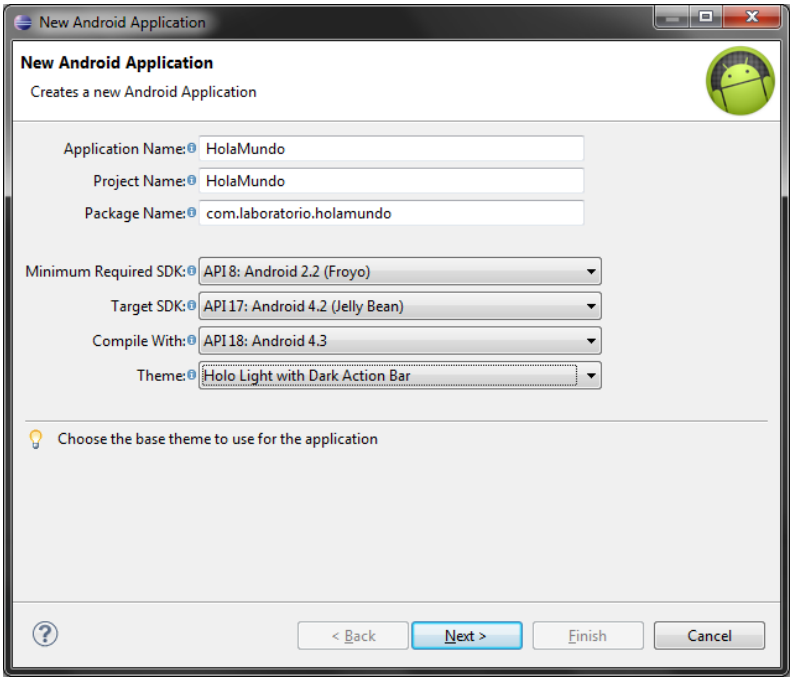


Ilustración 34. Creación del proyecto

En ella se elegirá el nombre de la aplicación, que será el nombre que tendrá la aplicación una vez instalada en el smartphone, el nombre del proyecto, que será el nombre que aparecerá en el workspace del Eclipse y el nombre de paquete, que es el identificador único que toda aplicación de Android ha de tener. Pulsando next se verá la siguiente pantalla:

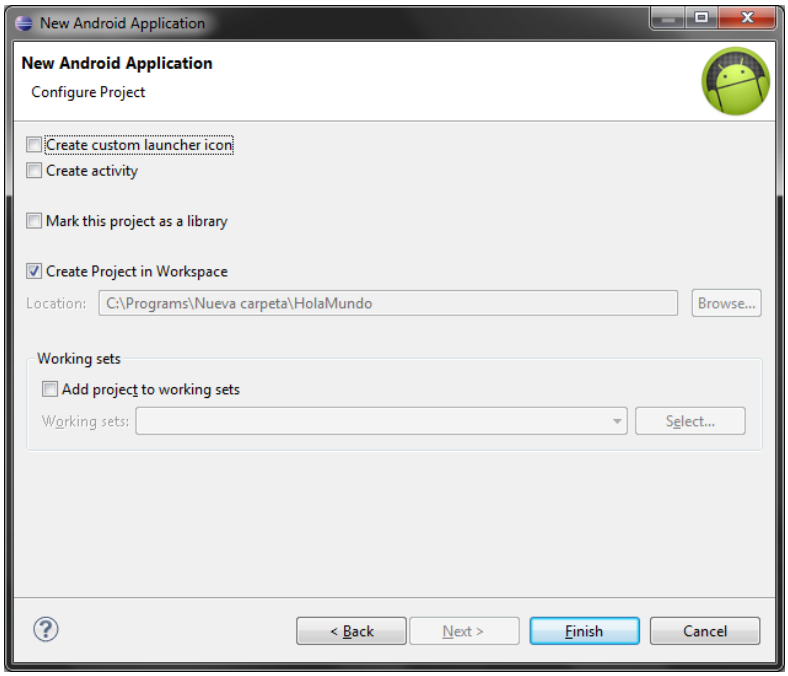


Ilustración 35. Opciones nueva aplicación Android

Aquí se puede elegir si creamos un icono personalizado para la aplicación y si se crea una Activity con una plantilla base. Para este caso se desmarcará esta casilla, para entender mejor los conceptos y crearla manualmente. Pulsando en Finish puede verse como en el workspace estác el proyecto que se acaba de crear. Si se despliega puede verse una serie de carpetas. De momento vamos a centrar la atención en la carpeta src y res.

En la carpeta src hay a crear el paquete que contendrá las actividades de la aplicación, para ello hay que pulsar sobre la carpeta con el botón derecho y elegir New -> Package. Como nombre habrá que elegir algo lógico para la ordenación de las clases, por ejemplo `com.laboratorio.holamundo.activities`

• 4.1.3 Creación de clases y layouts

Una vez creado el paquete, se pueden crear clases dentro de él. Para ello, de nuevo con el botón derecho sobre el nombre del paquete, New -> Class. Elegiremos un nombre para la Activity, como `HolaMundoActivity`. Se puede desde aquí elegir la superclase (Activity) que vamos a extender con esta clase o hacerlo directamente en el código.

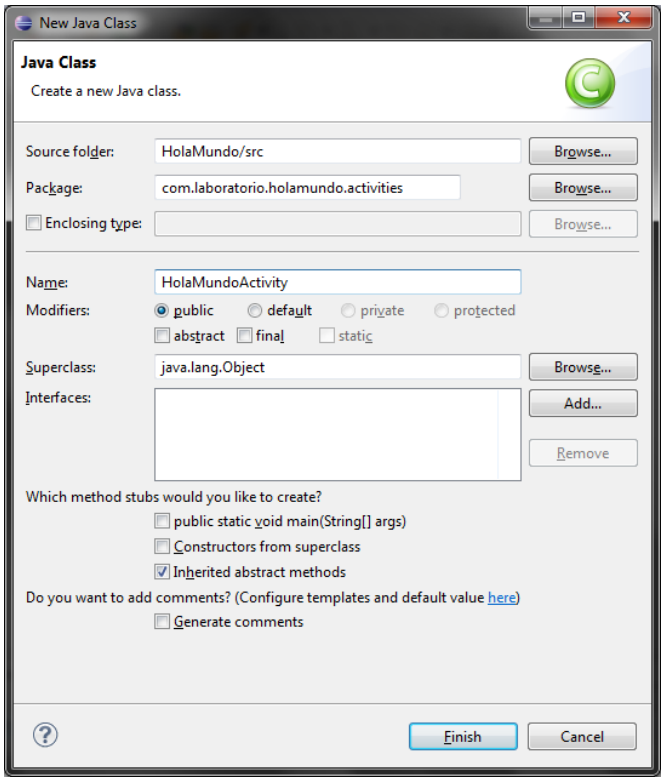


Ilustración 36. Nueva clase java

Cuando se pulsa Finish, la nueva clase es creada y podrá verse lo siguiente:

```
package com.laboratorio.holamundo.activities;

public class HolaMundoActivity {
}
```

Fragmento de código 10. Nueva clase creada

Ahora hay que hacer que esta clase extienda a Activity, para ello hay que escribir `extends Activity` e importar el paquete `android.app.Activity`.

```
package com.laboratorio.holamundo.activities;

import android.app.Activity;

public class HolaMundoActivity extends Activity {

}
```

Fragmento de código 11. Nueva Activity

Para comprender qué se va a hacer ahora, es importante tener presente el siguiente diagrama con el ciclo de vida de una actividad. En él vemos como el punto de inicio de una activity es el método OnCreate. Es imprescindible que se sobrescriba este método en la aplicación.

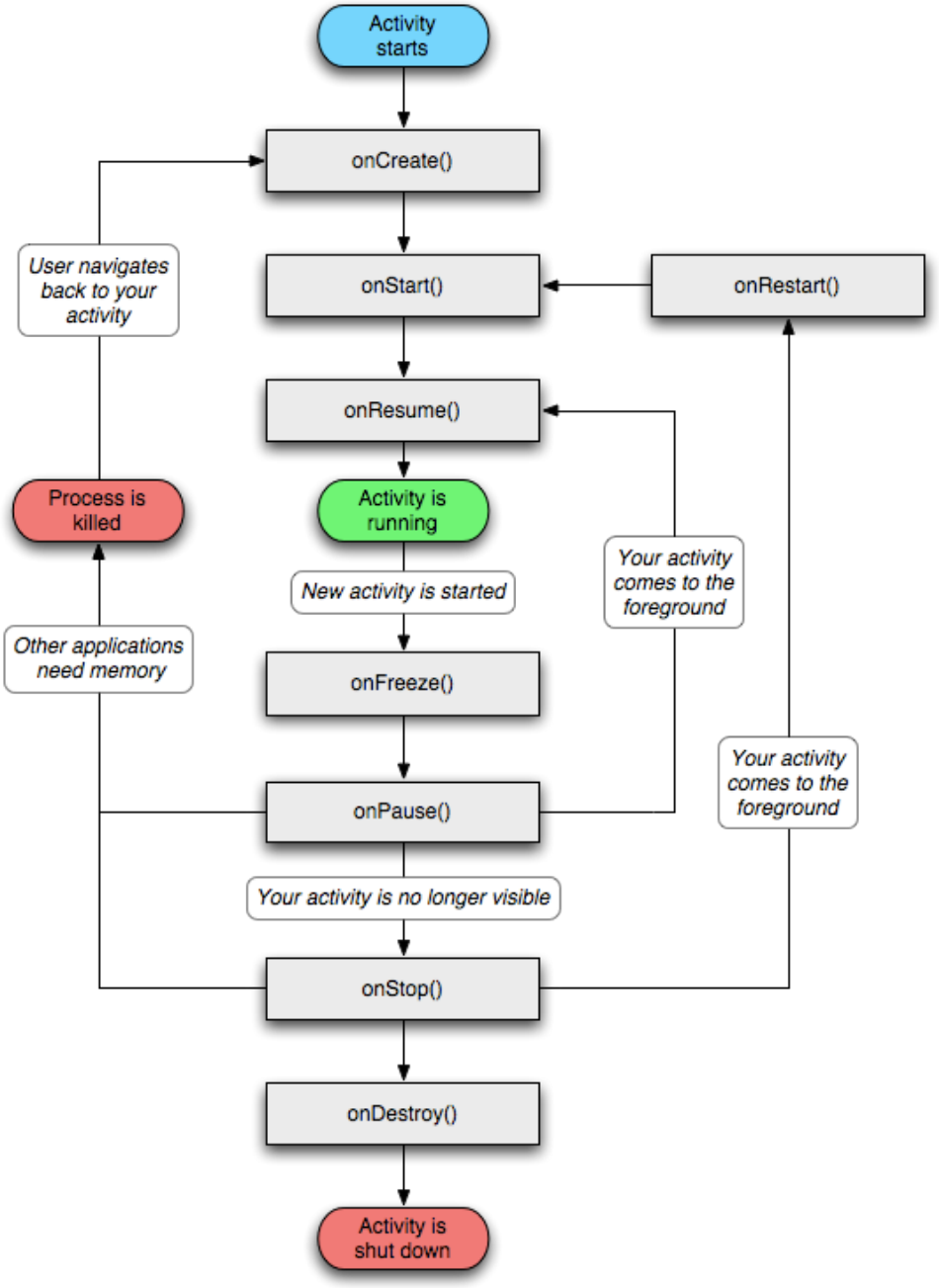


Ilustración 37. Ciclo de vida de una Activity

Atendiendo a esto, hay que sobrescribir el método onCreate:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
}
```

Fragmento de código 12. Método onCreate de una Activity

En este punto la actividad aún no es visible y está esperando que se le diga qué vista cargar para mostrar al usuario. Siguiendo un patrón Modelo-Vista-Controlador, el controlador sería la Activity,

mientras que la vista sería el layout que hay que crear. Para ello, pulsando con el botón derecho en la carpeta res/layout, New->Android XML File y aparecerá la siguiente pantalla:

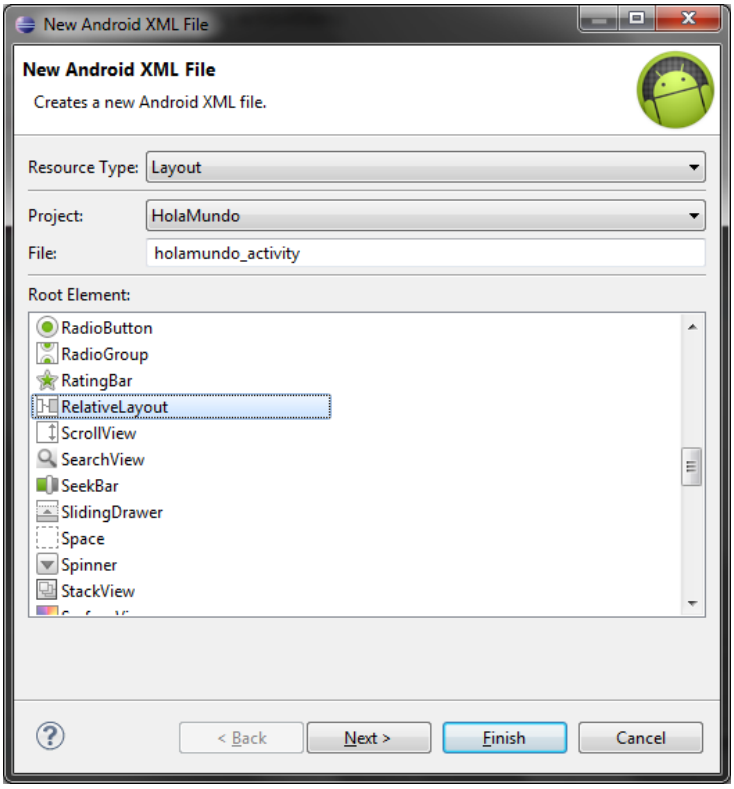


Ilustración 38. Nuevo Layout Android

Hay que elegir *layout* como Resource Type, y darle un nombre. El elemento raíz que se puede elegir aquí podría ser un LinearLayout o un RelativeLayout. La recomendación es usar el RelativeLayout, por la versatilidad que ofrece.

Cuando se pulse finish, una nueva pantalla se abre con una serie de herramientas para construir la interfaz gráfica. En la parte izquierda se puede ver la Paleta de widgets, desde la que se pueden arrastrar y soltar elementos gráficos. En la parte derecha se puede ver la estructura, que será muy útil para encontrar elementos una vez tengamos layouts complejos. En la parte de abajo vemos dos pestañas, una llamada Graphical Layout y otra holamundo_activity.xml. Aquí se podrá cambiar la vista desde xml a representación gráfica.

Es interesante destacar que el uso correcto de esta herramienta es muy importante para realizar una buena aplicación. El proceso de maquetación de las interfaces suele ser más costoso en tiempo que la propia programación e integración con sistemas externos. No debemos olvidar que Android tiene smartphones de todas las gamas y resoluciones, y hacer que las interfaces gráficas se vean bien en un porcentaje alto de modelos de móviles es muchas veces un reto.

En la siguiente captura puede verse lo que ha sido explicado previamente:

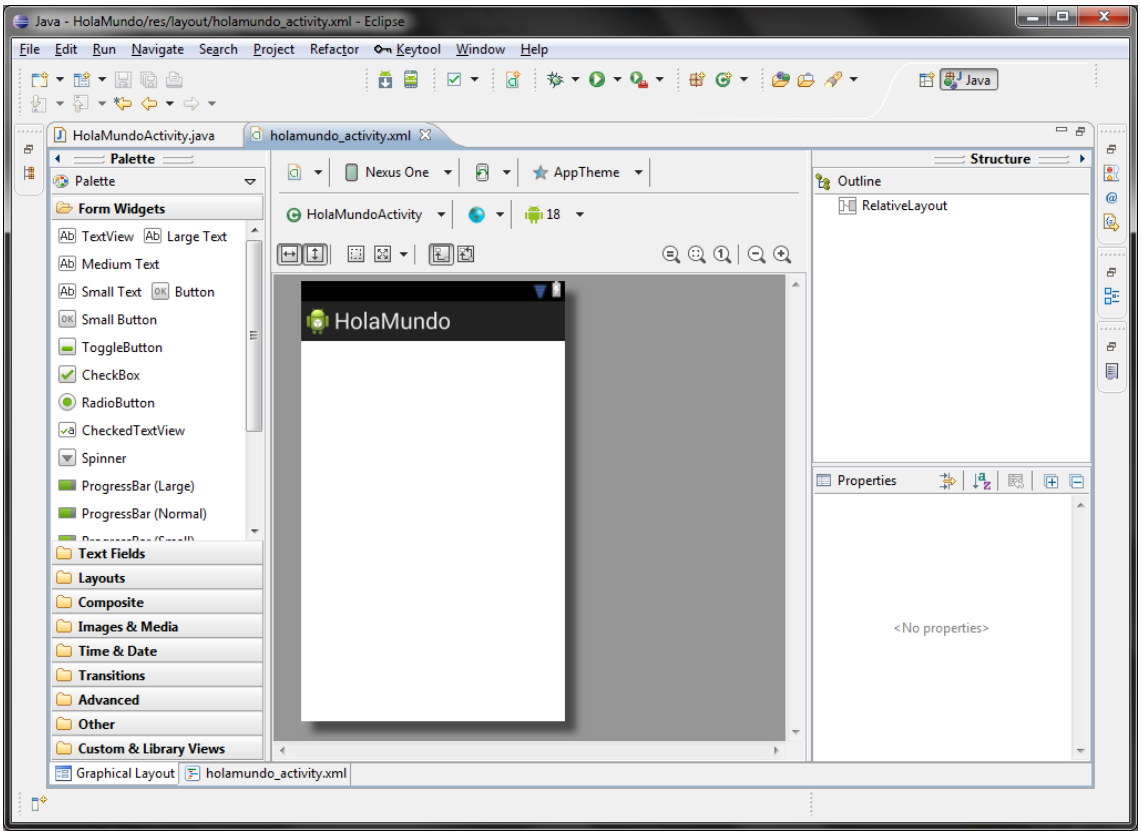


Ilustración 39. Plugin de dibujado de layouts

Ahora se arrastrará un Widget de tipo TextView a la pantalla. Si hemos elegido un RelativeLayout puede verse como aparecen líneas que nos van indicando como se va a colocar la vista dentro de la jerarquía. Se puede elegir pegarlo arriba, pegarlo a la izquierda, centrarlo horizontal/verticalmente, etc. Aquí cada alumno puede elegir dónde colocar la vista.

Una vez colocado el TextView, se pueden variar sus propiedades tales como tamaño de texto, color de texto, cuántas líneas ha de ocupar, etc… Un atributo imprescindible para un TextView es lógicamente el texto que contiene. La forma correcta de añadir un texto constante a un TextView es a través de referencias a la carpeta *values*. Tendrá que añadirse una nueva entrada al archivo *strings.xml* que se encuentra en la carpeta *res/values.* En este archivo se añadirá la siguiente línea:

```
<string name="hola_mundo">Hola, mundo!</string>
```

Fragmento de código 13. Nuevo recurso string

Una vez que se tenga esta cadena creada puede ser referenciada desde el TextView que se ha creado, para ello se editará el archivo *holamundo_activity.xml.* Ahora habrá que cambiar el elemento *android:text* del elemento TextView:

```
android:text="@string/hola_mundo" />
```

Fragmento de código 14. Asignación de recurso string a TextView

Si todo ha ido bien, podremos ver en la pre visualización de la vista, como ha cambiado el texto del textView.

Ilustración 40. Previsualización

Una vez creada la vista, solamente hace falta decirle a la Activity cuál es su layout correspondiente. Esto se hace añadiendo la siguiente línea de código al método onCreate.

```
setContentView(R.layout.holamundo_activity);
```

Fragmento de código 15. SetContentView

Aquí hay que darse cuenta de que la Activity no está en el mismo paquete que el archivo generado R.java, por lo que habrá que importar esta clase en HolaMundoActivity.

El último paso que queda para poder ejecutar la aplicación es declarar la Activity en el AndroidManifest. Para ello hay que añadir esto dentro del elemento application:

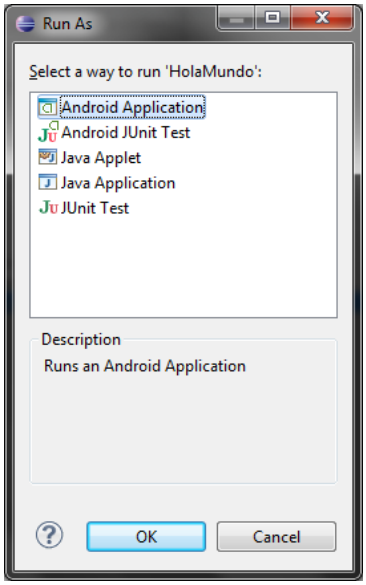
```
<activity
        android:name="com.laboratorio.holamundo.activities.HolaMundoActivity"
        android:label="@string/app_name" >
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
</activity>
```

Fragmento de código 16. Intent Filter para launcher

Con el intent-filter estamos diciendo que esa actividad es lanzadera de aplicación.

• 4.1.4 Ejecución de la aplicación

El proyecto está en disposición de ser ejecutado en móvil o en emulador. Para ello hay que pulsar en el icono de iniciar aplicación  . Saldrá un pop up igual a este:



Fragmento de código 17. Ejecución de la aplicación

Hay que elegir Android application. Podrá verse después de dar a OK como la aplicación se inicia en el emulador/móvil

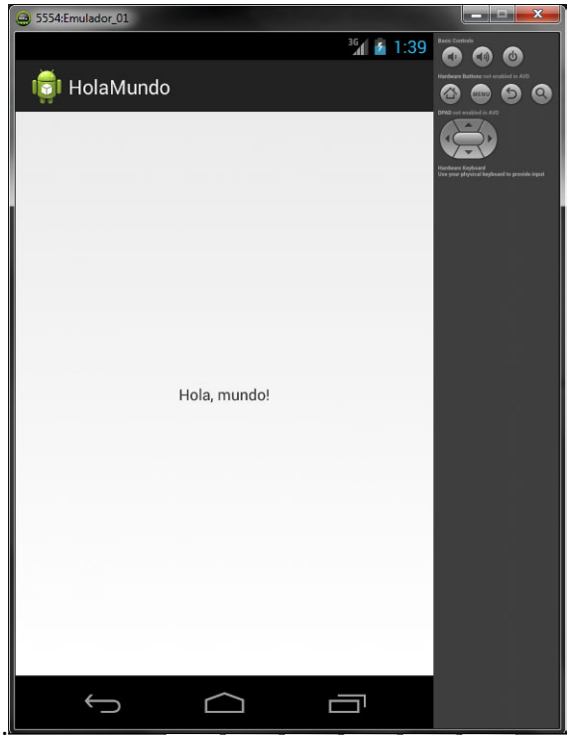


Ilustración 41. Hola mundo corriendo

• 4.1.5 Trabajo posterior

Un buen ejercicio para comprender el ciclo de vida de una actividad es la clase `Toast`. Esta clase sirve para mostrar notificaciones al usuario en forma de una ventana modal que se muestra y oculta automáticamente. La idea es añadir esta línea de código a cada uno de los siguientes métodos:

- onCreate
- onResume
- onPause
- onDestroy

De esta manera podrá verse más claramente como la aplicación pasa de un estado a otro según está explicado en el diagrama de ciclo de vida.

Para ello habrá que añadir esto a cada uno de estos métodos:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
	setContentView(R.layout.holamundo_activity);
	Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
}

@Override
protected void onResume() {
	super.onResume();
	Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
}

@Override
protected void onPause() {
	super.onPause();
	Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
}

@Override
protected void onDestroy() {
	super.onDestroy();
	Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
}
```

Fragmento de código 18. Ciclo de vida con Toasts

Si ahora se inicia la aplicación, puede apreciarse como aparecen las notificaciones según la actividad va pasando por cada uno de los estados del ciclo de vida.

Con la aplicación iniciada el alumno deberá probar a minimizarla pulsando el botón Home y finalizarla pulsando el botón Back. De esta manera podrá apreciarse el paso por cada uno de los métodos para así afianzar el ciclo de vida, que es primordial para controlar la aplicación en todos sus estados.

4.2 Práctica demostrativa 1. Desarrollo de interfaces de usuario básicas con Android.

Los objetivos de esta práctica son componer una interfaz básica de usuario que incluya algunos de los widgets más importantes que se pueden encontrar en una aplicación Android. Entre ellos se prestará especial atención al componente *ListView*.

Esta vista se representa en pantalla como una lista con desplazamiento vertical que es capaz de mostrar los elementos del `Adapter` que tenga asociado. El formato con el que se muestran los elementos puede ser modificado pudiendo incluir cualquier componente gráfico que extienda de la clase `View` dentro de cada uno de ellos.

El Adapter *traduce* cada uno de los ítems según un modelo de datos al layout establecido. Este layout establecido es cada una de las filas de la lista y puede ser tan complejo como se necesite. En este caso vamos a crear una estructura como la de la siguiente imagen:

Ilustración 42. Layout para lista

Para este ejemplo se utilizarán las fases de la luna como modelo de datos. Son ocho las fases de la luna:

- Luna nueva o novilunio
- Creciente iluminante
- Primer cuarto o cuarto creciente
- Gibosa iluminante
- Luna llena o plenilunio
- Gibosa menguante
- Último cuarto o cuarto menguante
- Creciente menguante

Facilitaremos los iconos para cada una de ellas:

Fases lunares.zip

• 4.2.1 Objetivos de la práctica

La aplicación Android ha de implementar una lista que ha de mostrar las ocho fases de la luna con su icono, nombre y nombre alternativo si lo tiene. A la lista hay que asociarla un *adapter*. Existen numerosas subclases que implementan la interfaz Adapter. La recomendación es extender la clase *BaseAdapter* para entender correctamente lo que se consigue con un adapter bien implementado.

Hay otras clases que abstraen más el funcionamiento interno y son más fácilmente integrables, pero insistimos en no usarlas, al menos al principio.

• 4.2.2 Modelo de datos

Para seguir correctamente el patrón Modelo-Vista-Controlador, tenemos actualmente identificadas dos partes (Vista = ListView; Controlador = Adapter). El modelo en este caso serían las fases lunares. Cada una de las fases lunares tiene los siguientes atributos:

- Imagen: Los archivos png en la carpeta *res,* se referencian en la aplicación como un entero.
- Nombre: Una cadena de caracteres representando el nombre de la fase lunar
- Nombre alternativo: Otra cadena de caracteres representando el nombre alternativo de la fase lunar, si lo tiene.
- Detalle: Cadena de caracteres con el detalle explicativo de la fase.

Es muy importante ser capaz de extraer e identificar el modelo de datos de unos requisitos funcionales de una aplicación. En este caso es muy sencillo, pero normalmente hace falta dedicarle un tiempo importante de análisis. Un ejemplo de posible implementación sería este:

```
public class MoonPhase {

        private int imageResource;
        private String name;
        private String altName;
        private String description;

        /**
         * Constructor
         * @param imgRes Recurso de la imagen
         * @param n nombre de la fase
         * @param altN nombre alternativo de la fase
         * @param desc descripción de la fase
         */
        public MoonPhase(int imgRes, String n, String altN, String desc){
                imageResource = imgRes;
                name = n;
                altName = altN;
                description = desc;
        }

        /**
         * Resto de getters y setters
         */
}
```

Fragmento de código 19. Modelo de datos FaseLunar

• 4.2.3 BaseAdapter

Se explicará brevemente qué ha de hacer un BaseAdapter, y los métodos que hay que sobreescribir. El adapter ha de tener acceso al modelo de datos, por lo que parece lógico que en su constructor haya que pasárselo. Hay cuatro métodos que necesariamente habrá que que implementar:

- *getCount:* Este método ha de devolver el número total de objetos que se van a representar.
- *getItem (int position):* Este método ha de devolver el objeto que ocupa la posición *position* en la lista
- *getItemId (int position):* Este método ha de devolver el identificador del objeto, en el caso de que tenga. Este caso no aplica, sin embargo debemos implementar el método. Será suficiente que se escriba un *return 0;*
- *getView (int position, View convertView, ViewGroup parent):* Este método va a devolver cada uno de los items de la lista. Es en este método donde se hará el ***inflate*** de la vista y se rellenarán sus campos en base a la *position* que sea. El parentView simplemente es la vista padre a la que se añadirá esta vista. La vista convertView merece una mención especial para comprender qué representa y está explicado en el punto cuatro de esta práctica.

Se ha hablado de hacer el ***inflate*** de la vista. Este proceso simplemente significa que desde un archivo XML vamos a generar un objeto View. El archivo XML será el que hayamos generado para cada uno de los elementos fase lunar en el primer apartado. Para esta tarea Android nos proporciona un objeto del sistema llamado *LayoutInflater,* que genera un objeto View desde un recurso layout.

Para conseguir el objeto LayoutInflater, hace falta el contexto de la aplicación, por lo que al BaseAdapter habrá que pasarle el *Context* como parámetro. Un ejemplo del constructor del BaseAdapter podría ser este:

```
private ArrayList<MoonPhase> items;
private Context mContext;
private LayoutInflater mInflater;

public MoonPhasesAdapter(ArrayList<MoonPhase> mList, Context ctx)
{
    items = mList;
    mContext = ctx;
    mInflater = (LayoutInflater) mContext.
                            getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}
```

Fragmento de código 20. Constructor de MoonPhasesAdapter

Atendiendo a esto, el método getView deberá crear cada vista y rellenar cada uno de los campos según el modelo de datos. Para referenciar cada uno de los campos habrá que usar el método *findViewById(int id)* de la clase View.

Una vez implementado el Adapter y asociado a la ListView deberíamos ver algo por el estilo:

Ilustración 43. ListView corriendo

• 4.2.4 ConvertView y uso de ViewHolders

Como se ha mencionado antes, el método *getView* de la clase *BaseAdapter* tiene un parámetro de entrada llamado *View convertView.* Un ListView normalmente contiene más elementos que los que muestra en la pantalla. Si el usuario hace *scroll*, los elementos se ocultan por la parte de arriba de la pantalla. Los objetos vista que desaparecen pueden ser reutilizados por los elementos que están a punto de aparecer por la parte de debajo de la pantalla.

Si Android determina que una vista que representa un elemento de la vista ya no es visible, le permite al método getView del adapter a reutilizarla mediante el parámetro convertView. De esta manera no hay que instanciar un nuevo objeto cada vez que el método getView es invocado. En el caso de que no haya vistas reutilizables el parámetro convertView es null. La implementación del adapter tiene que chequear esto.

El patron ViewHolder evita que usemos el método *findViewById* en una *convertView* reutilizada. Un ViewHolder es una clase estática interna al adapter que mantiene referencias a las vistas relevantes del layout que estemos utilizando: en este caso la imagen, el nombre y el nombre alternativo. Estas referencias las asignamos al objeto vista que representa el elemento de la lista mediente el método *setTag().*

Así, cuando recibimos un objeto convertView, se puede obtener su ViewHolder mediante el método *getTag()*, y se puede asignar directamente, sin realizar más findViewById, los valores que procedan según esa posición.

Puede sonar complejo, pero este patrón hace que ganemos un 15% de velocidad en la ejecución del método getView. Se verá más claro con un ejemplo:

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {

	FakeObject nObject = items.get(position);
	ViewHolder holder;

	//Si converView es null, no tenemos referencias, por lo que habrá que crear un objeto ViewHolder y asignárselo a esta vista
	if(convertView == null)
	{
		convertView = mInflater.inflate(R.layout.fase_lunar_item, null);

		holder = new ViewHolder();
		holder.fakeName = (TextView)convertView.findViewById(R.id.name);

		convertView.setTag(holder);
	}
	//En cambio si convertView no es null, sabemos que viene un objeto ViewHolder con referencias a vistas hijas relevantes
	else
	{
		holder = (ViewHolder) convertView.getTag();
	}

	//En este punto las referencias las tenemos seguro
	holder.fakeName.setText(nObject.getName());

	return convertView;
}


private static class ViewHolder{

	public TextView fakeName;
}
```

Fragmento de código 21. Patrón ViewHolder

• 4.2.5 Uso de varias Actividades en una misma aplicación

El uso de varias actividades es una práctica común en cualquier aplicación Android. La idea en esta práctica es hacer otra Activity para mostrar el detalle cuando se pulse en un elemento de la lista. El detalle consistirá en un simple *TextView* con información ampliada explicando la fase lunar. Estos detalles serán los siguientes:

- Luna Nueva o Novilunio, también llamada "Luna Nueva Astronómica" o "Luna Negra", corresponde a la Luna Nueva Verdadera; esta fase de la Luna normalmente es imposible verla a simple vista ya que se encuentra oculta tras el resplandor solar, sólo es posible observarla cuando ocurre un eclipse total de Sol, los cuales acontecen durante esta fase lunar sólo cuando las condiciones dadas son las adecuadas.
- Luna Creciente, también llamada Nueva Visible, corresponde a la Luna Nueva Tradicional y es la primera aparición de la Luna en el cielo, 18 o 30 horas después de haberse producido la posición de "Luna Nueva Astronómica". Esta fase de la Luna se podrá ver en el cielo hacia el oeste, una vez ya ocultado el Sol, justo por encima del crepúsculo aún restante. Tiene forma de pequeña guadaña o cuerno. Esta fase de la Luna es la que se utiliza para dar comienzo al primer día de cada mes lunar.
- Cuarto Creciente. Tiene su salida en el horizonte por el este a las 12 del mediodía (hora astronómica local, no necesariamente hora oficial), su cenit se produce a las 6 de la tarde y su ocaso a las 12 de la medianoche. La parte luminosa de la Luna durante esta fase tiene la forma de un círculo partido justo a la mitad (semi-círculo).
- Luna Gibosa Creciente, una vez ya pasada la fase del Cuarto Creciente, la Luna va tomando progresivamente día tras día, una forma convexa por ambos lados en su parte luminosa, perdiendo ese lado recto que poseía durante la fase Cuarto Creciente.
- Luna Llena o Plenilunio, es cuando la concavidad de la parte luminosa de la Luna se logra completar en su totalidad hasta formar un círculo. Su salida en el horizonte es aproximadamente a las 6:00 p.m., el cenit lo alcanza aproximadamente durante la medianoche y se oculta cerca de las 6:00 de la mañana. La Luna Llena viene a marcar justo lo que es la mitad del mes lunar (14 días, 18 horas, 21 minutos 36 segundos).
- Luna Gibosa Menguante, pasada ya la fase correspondiente a la Luna Llena, la parte luminosa de la Luna comenzará a menguar con el correr de los días, tomando así de nuevo —igual como en la Luna Gibosa creciente— una apariencia de una Luna-Cóncava (gibosa) esta vez en su fase decreciente.
- Cuarto Menguante, exactamente igual que el Cuarto Creciente, pero en sentido contrario. Además, tiene su salida en el horizonte a las 12 de la medianoche, alcanza el cenit en el cielo a las 6 de la mañana y su ocaso se produce a las 12 del mediodía, es decir, esta fase lunar corresponde al periodo de días durante el cual es posible observar a la Luna en el cielo durante las horas de la mañana.
- Luna Menguante, conocida también como "Creciente Menguante" o "Luna Vieja" (este último término poco conocido) ya que es idéntica a la Luna Nueva Visible, pero en sentido opuesto. La Luna Menguante sólo es posible verla de madrugada, hacia el Este, justo por encima de la Aurora o Alba y antes de que salga el Sol. Tiene apariencia de pequeña guadaña.

Para iniciar una nueva actividad se usará el objeto *Intent.* Son descripciones abstractas de operaciones que se van a ejecutar. En este caso la operación a ejecutar va a ser la de abrir una nueva *Activity,* por lo que la sintaxis que habrá que usar será la siguiente:

```
Intent newActivity = new Intent(getApplicationContext(), ActivityDestino.class);
```

Fragmento de código 22. Creación de nuevo Intent

Siendo ActivityDestino la *Activity* que se quiere iniciar.

Los *Intents* pueden llevar información adicional mediante el método *putExtra*. Se recomienda estudiar la información de las API para ver todas las sobrecargas de este método y los parámetros que acepta. En este caso, dado que la única información que necesita la actividad de detalle es una cadena de caracteres, se puede usar el método *putExtra(String key, String value)*.

En el método onCreate de la actividad detalle hay que obtener este parámetro. La forma de hacerlo es la siguiente:

```
Intent callingIntent = getIntent();
Bundle extras = callingIntent.getExtras();
String descripcion = extras.getString("detalleFase");
```

Fragmento de código 23. Paso de parámetros entre activities

Una vez se tenga el parámetro habrá que mostrarlo en un TextView. Antes de arrancar la aplicación hay que declarar la nueva Activity en el manifest. El resultado final será algo parecido a esto:

Ilustración 44. Detalle fase lunar

4.3 Práctica demostrativa 2. Interfaces avanzadas y fragmentos

En esta práctica se abordará la construcción de interfaces más complejas y atractivas para el usuario final de la aplicación. En concreto se usará una estructura de menú lateral y Fragmentos intercambiables en una FragmentActivity.

Para ponernos en contexto, una FragmentActivity es una clase que extiende a Activity, por lo que respeta el mismo ciclo de vida estudiado en la Práctica 0. A parte de esto, tiene un componente esencial para el uso de Fragments y es el FragmentManager. Dicho componente sirve para añadir, quitar y reemplazar fragmentos, entre otras cosas.

También se usará, para el menú lateral, la clase NavigationDrawer. Vemos un ejemplo de uso:

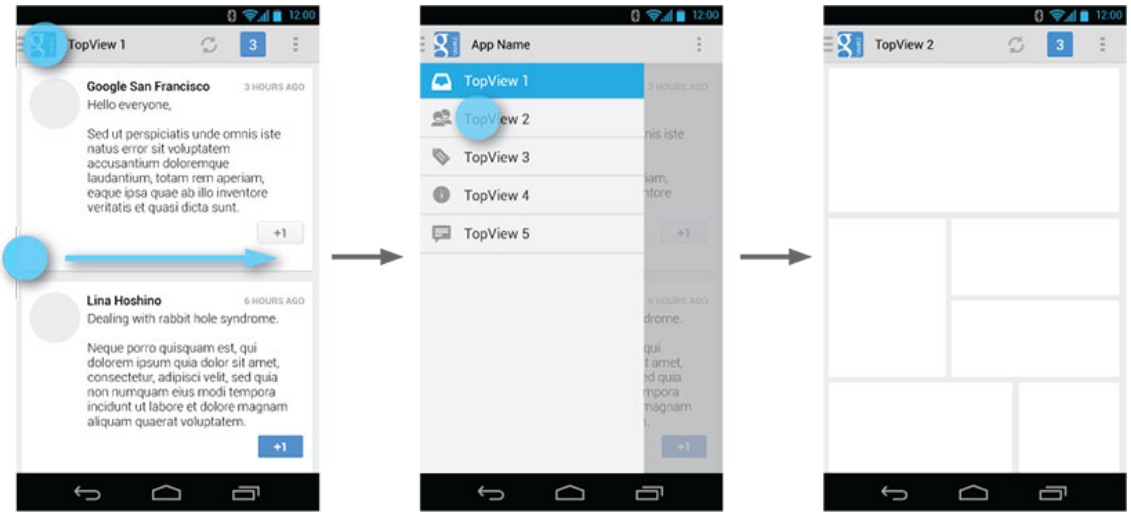


Ilustración 45. Navigation Drawer

• 4.3.1 Instalación de Librería de Compatibilidad

NavigationDrawer es una clase está incluída en la librería de soporte de Android, que puede ser instalada pulsando con el botón derecho en el proyecto → Android tools → Add Support Library de la siguiente manera:

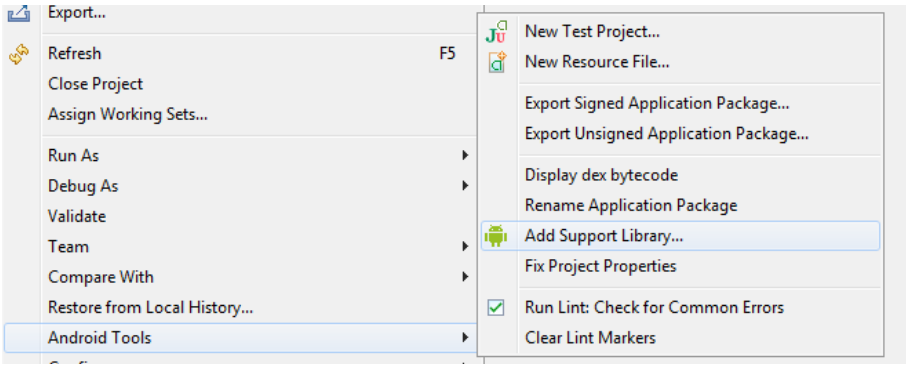


Ilustración 46. Librería de soporte

Aparecerá la siguiente pantalla:

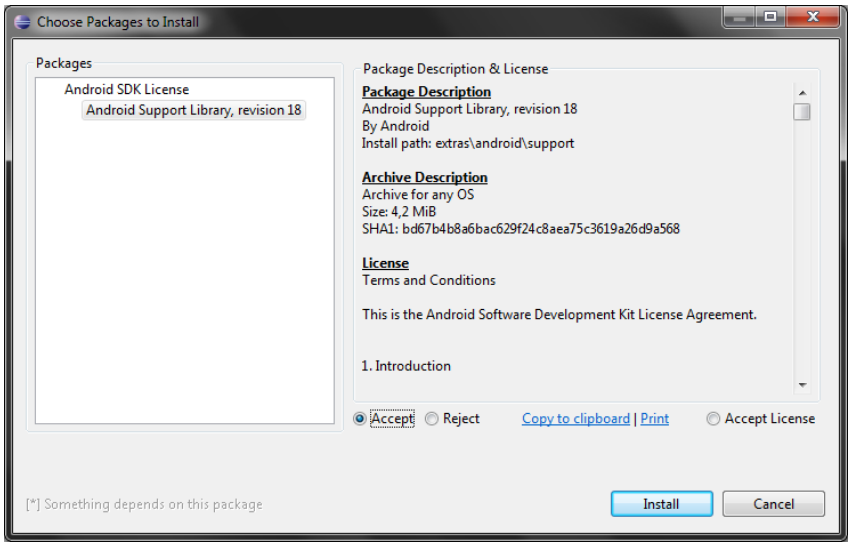


Ilustración 47. Licencia de librería de soporte

Después de descargar e instalar la librería, en el desplegable Android Private Libraries debería aparecer un archivo .jar llamado **android-support-v4.jar**.

• 4.3.2 Creación del NavigationDrawer

Una vez instalada la librería de compatibilidad se puede crear el layout que va a contener la estructura de menú lateral y una vista (en este caso un FrameLayout) que contendrá los diferentes Fragments.

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Vista que va a contener todos los Fragments -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <!-- Navigation Drawer -->
    <ListView
        android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice" />

</android.support.v4.widget.DrawerLayout>
```

Fragmento de código 24. Layout de NavigationDrawer

Para la correcta colocación hay que respetar estas reglas, tomadas directamente de developer.android.com:

- La vista principal (en este caso el FrameLayout) ha de ser el primer el primer element hijo del DrawerLayout porque el orden en los hijos en el XML implica el *z-order,* y la lista ha de estar en primer plano.
- La vista del Drawer (la ListView) debe especificar su atributo layout_gravity. En este caso los valores “left” o “start” harían que el contenido de la lista apareciera en la izquierda.
- La vista Drawer especifica su ancho en unidades dp. No debe ser más ancha que 320 dp para que no cubra la totalidad del contenido principal.

• 4.3.3 FragmentManager y operaciones sobre Fragmentos

Una vez que el usuario pulse en cada uno de los ítems de la lista del NavigationDrawer, el contenido de la Actividad ha de cambiar. La forma correcta de hacer este cambio es usando Fragments con los diferentes layouts y cambiarlos usando el objeto FragmentManager. Para tener acceso a dicho objeto, como se ha dicho en el punto uno, la Activity tiene que ser una FragmentActivity.

Se pide, pues, que el alumno implemente un *OnItemClickListener* en la lista lateral del NavigationDrawer y cambie los Fragmentos de la Activity principal mediante el método **replace** del objeto FragmentManager.

Los fragmentos a implementar han de ser dos: un fragmento que presente un layout de lista, como en la práctica anterior, y otro fragmento utilizando el widget GridView de dos columnas para la presentación de los datos. Hay que recordar que el adapter puede ser el mismo al utilizado en la práctica anterior. Únicamente hay que tener en cuenta que el layout de cada ítem del GridView ha de ser diferente al del ListView, como se muestra en la siguiente captura:

Ilustración 48. Layout de item para GridView

• 4.3.4 Fragmento con WebView

Se añadirá un tercer fragmento a la lista del NavigationDrawer. Se pide que el alumno implemente un Fragmento cuya vista sea un WebView que cargue la URL http://m.es.wikipedia.org/wiki/Fase_lunar.

Cabe recordar aquí la necesidad de añadir el permiso de acceso a internet en el AndroidManifest.

A continuación se mostrarán algunas capturas de pantalla de cómo deberá verse la aplicación una vez terminada.

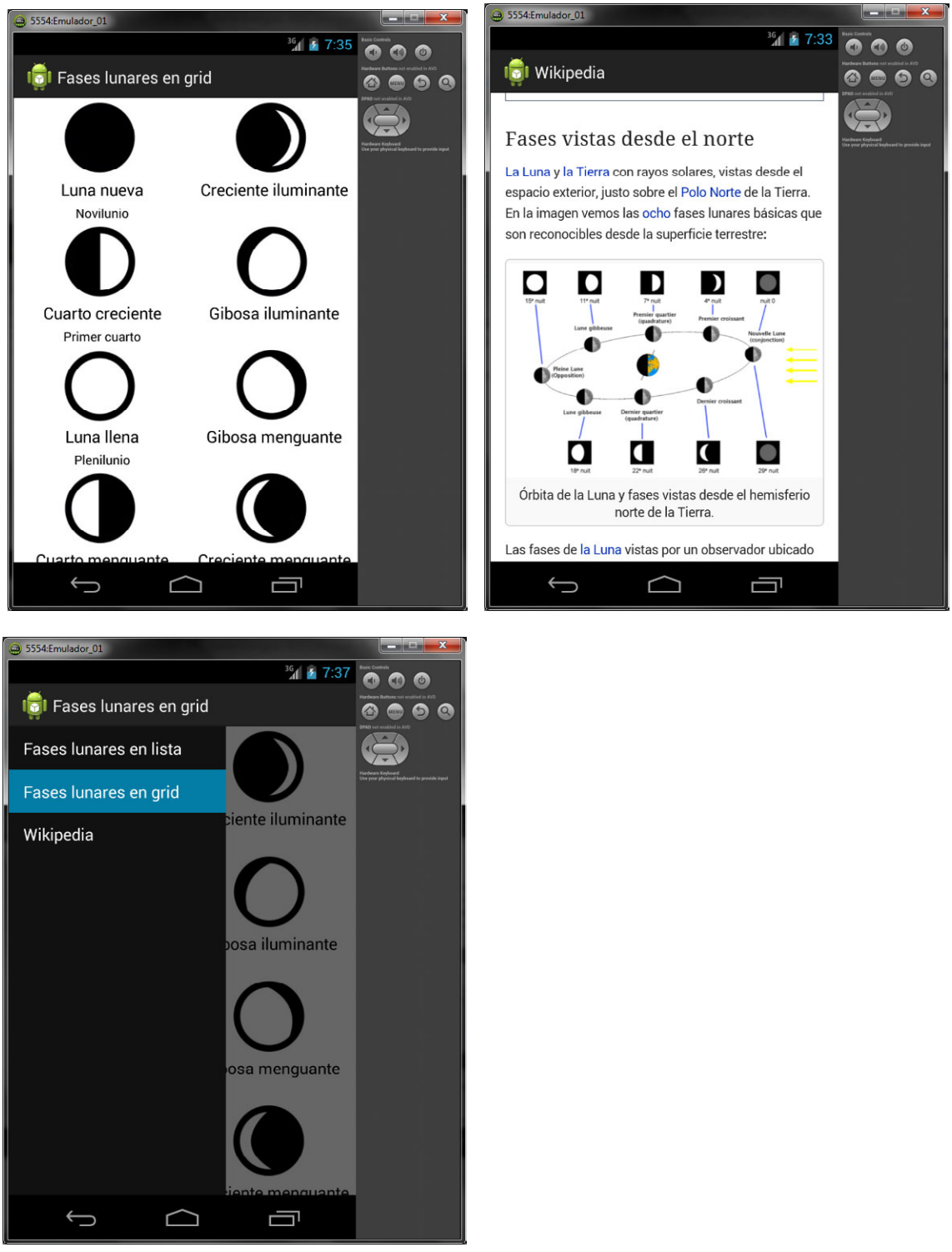


Ilustración 49. Capturas práctica 2

4.4 Práctica demostrativa 3. Comunicaciones y consultas asíncronas con AsyncTask

El objetivo de esta práctica es que el alumno se familiarice con las consultas de archivos y contenidos remotos mediante HTTP. Las aplicaciones son la capa de presentación de los sistemas, y muchas veces los servicios sobre los que se apoyan no son locales. Lógicamente estas llamadas a servicios externos no son rápidas, y pueden o no estar disponibles, por lo que a la hora de desarrollar una aplicación habrá que tener esto muy en cuenta.

Si una aplicación no responde mientras está haciendo una consulta HTTP, o algún acceso a base de datos, trabajo con fichero, etc, la usabilidad se ve afectada y la percepción del usuario es negativa. Como estas consultas bloquean la interfaz de usuario, Android posee clases de ayuda para realizar estas tareas de manera asincrónica. En concreto centraremos el estudio en la clase AsyncTask.

• 4.4.1 Clase AsyncTask

Como se ha explicado la clase AsyncTask sirve para realizar trabajos que bloquean la interfaz de usuario en segundo plano. Está pensada para ser implementada dentro de otra clase y al ser abstracta, siempre tendremos que extenderla. Un AsyncTask está definido por tres tipos genéricos, llamados *Params, Progress* y *Result,* y cuatro pasos, que son *onPreExecute, doInBackground, onProgressUpdate* y *onPostExecute.*

El único método que hay que implementar obligatoriamente es el *doInBackground*, los otros tres son opcionales. Normalmente se implementa también el método *onPostExecute*, para realizar acciones con el o los objetos que se han obtenido en el *doInBackground.*

Los tres tipos usados por un AsyncTask son los siguientes:

- Params, que es el tipo de parámetro de entrada enviado en el momento de la ejecución
- Progress, el tipo de unidad que se usará para mostrar el progreso en la intefaz principal
- Result, que es el tipo del resultado después de hacer el trabajo en segundo plano.

No todos los tipos son usados por un AsyncTask. Para marcar un tipo como no usado, simplemente hay que usar el tipo *Void.*

```
private class DownloadCamerasInBackground extends AsyncTask<Void, Void, Void>
```

Cuando un AsyncTask es ejecutado para por estos cuatro pasos:

- ***onPreExecute****:* Es invocado en el hilo principal **antes** de que el task sea ejecutado. Normalmente se usa para actualizar la vista principal ya que desde *doInBackground* no está permitido ya que la vista solo puede cambiarse desde su mismo thread. Usualmente se puede poner aquí un ProgressDialog para indicar una actividad en segundo plano.
- ***doInBackground****:* Justo después de que termina la ejecución del método *onPreExecute*, el *doInBackground* es ejecutado en un thread diferente al de la interfaz principal. Aquí es

donde pondremos el trabajo que bloquea la interfaz. Si queremos actualizar la vista principal desde aquí, habría que llamar al método *publishProgress*.

- ***onProgressUpdate:*** Este método es llamado cada vez que se usa el método publishProgress desde el doInBackground. Como este método es ejecutado en el hilo principal de la interfaz gráfica, se puede hacer cambios en ésta, ya sea actualizando una barra de progreso, o simplemente mostrando un *Toast.*
- ***onPostExecute:*** Este método es llamado en el hilo principal cuando doInBackground termina su ejecución.

• 4.4.2 Conexiones HTTP

Android ofrece multitud de clases para realizar las conexiones a servicios remotos. Muchas veces lo difícil es saber qué clase elegir para los objetivos establecidos. Se recomienda estudiar las posibilidades diferentes que ofrecen las clases *android.net.http.AndroidHttpClient* y *java.net.URL*.

La finalidad con ambas opciones es obtener un **InputStream** de la conexión que se pueda transformar en el tipo de objeto que se esté buscando, ya sea una cadena de caracteres, una imagen, etc. En este caso, de este InputStream necesitamos sacar un objeto **Bitmap**. La forma más sencilla es usar el objeto BitmapFactory y su método decodeStream que toma como parámetro de entrada un InputStream. Una vez se tenga el Bitmap puede ser vinculado al ImageView mediante el método setImageBitmap.

Para esta práctica se usará la estructura de la práctica 2. Una estructura de menú lateral con fragmentos es sumamente escalable, como veremos ahora. La idea es que se cree una nueva entrada en el menú lateral que haga referencia a un nuevo fragmento que contenga cuatro imágenes que se carguen dinámicamente desde internet.

En concreto las imágenes van a ser de las cámaras del ayuntamiento de Madrid: http://informo.munimadrid.es/informo/Camaras/. Dejamos al alumno que elija arbitrariamente cuatro archivos jpg para realizar la práctica. El layout final podrá ser algo parecido a estas capturas.

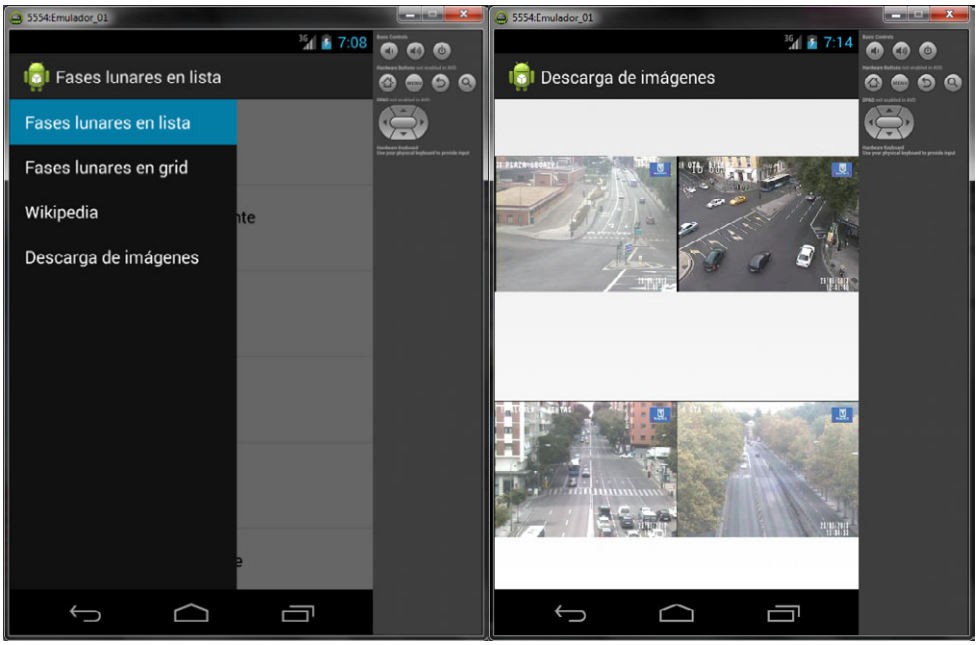


Ilustración 50. Capturas práctica 3

• 4.4.3 Conclusiones

En esta práctica se ha visto cómo usar un AsyncTask para descargar archivos de internet. Sin embargo un AsyncTask ha de ser usado siempre que un proceso bloquee la interfaz de usuario. Puede ser que estemos intentando acceder a una base de datos, o haciendo trabajos con ficheros, o incluso cálculos matemáticos pesados.

La usabilidad de una aplicación es algo que muchas veces no es posible medir, pero los usuarios sí la perciben. Es importante realizar estas acciones para que esta percepción del usuario sea que, aunque la aplicación tenga que hacer tareas en background, la interfaz siempre responda.

4.5 almacenamiento de datos. SharedPreferences y SQLite

En esta práctica se abordará el almacenamiento de datos en una aplicación Android. Existen varias formas de para almacenar información en un sistema Android. Cada una de ellas tiene sus características, ventajas, e inconvenientes.

En cuestión vamos a centrarnos en el uso de SharedPreferences y base de datos SQLite. Se ha dejado fuera de esta práctica el almacenamiento de archivos porque no difiere en demasía de la forma en la que accedemos a estos servicios en Java.

• 4.5.1 Almacenamiento en SharedPreferences

El almacenamiento en SharedPreferences se usa comúnmente para guardar datos de usuario, tales como preferencias de configuración o el estado de la aplicación cuando el usuario la cierra.

Para esta parte de la práctica se pide que el alumno implemente un layout sencillo con cuatro CheckBoxes simulando las opciones de configuración de una aplicación. Algo como esta captura:

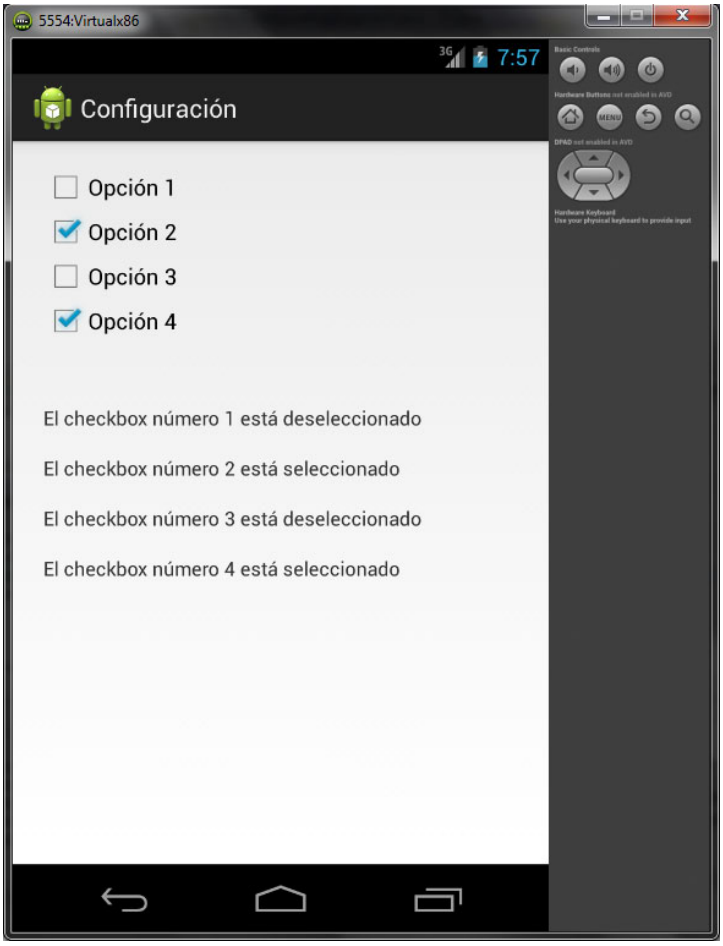


Ilustración 51. Layout de ejemplo para SharedPreferences

El funcionamiento requerido es que cuando el usuario abandone la aplicación los CheckBoxes seleccionados/no seleccionados se queden guardados en el espacio proporcionado por SharedPreferences. Se recomienda estudiar las posibilidades ofrecidas por este objeto para salvar/recuperar datos guardados.

• 4.5.2 Almacenamiento en base de datos SQLite

Como se ha visto en el punto anterior, para guardar pequeñas cantidades de datos de tipos primitivos, la clase SharedPreferences es de gran utilidad. Sin embargo es normal que en una aplicación mediana, haya datos que tengamos que almacenar para que sean persistentes entre sesiones para mejorar tiempos de carga, ahorrar datos descargados, etc.

Android ofrece una serie de clases para montar una base de datos SQLite en el almacenamiento privado de la aplicación. Atención especial hay que prestarle a la clase SQLiteOpenHelper. Es una clase abstracta por lo que habrá que extenderla y sobreescribir los siguientes métodos:

- OnCreate: Aquí es donde hay que crear las tablas de la base de datos. Es únicamente llamado cuando la base de datos es creada.
- OnUpdate: Este método es llamado cuando la base de datos cambia de versión al haberse visto modificada su estructura.

Para esta parte de la práctica se pide crear una base de datos con una tabla acorde al modelo de datos de la práctica 1. Lo recordamos aquí:

```
public class MoonPhase {

        private int imageResource;
        private String name;
        private String altName;
        private String description;

        /**
         * Constructor
         * @param imgRes Recurso de la imagen
         * @param n nombre de la fase
         * @param altN nombre alternativo de la fase
         * @param desc descripción de la fase
         */
        public MoonPhase(int imgRes, String n, String altN, String desc){
                imageResource = imgRes;
                name = n;
                altName = altN;
                description = desc;
        }

        /**
         * Resto de getters y setters
         */
}
```

Fragmento de código 25. Modelo de datos FaseLunar

Se pide que el alumno implemente la lógica necesaria para almacenar los datos provistos en la práctica uno así como los procedimientos necesarios para recuperar estos datos almacenados y poblar la lista con las fases lunares.

A modo de ayuda, a continuación se muestra parte del código de una clase que extiende a SQLiteOpenHelper.

```
public class DatabaseHandler extends SQLiteOpenHelper {

	// Versión de la base de datos
    private static final int DATABASE_VERSION = 1;

    // Nombre de la base de datos
    private static final String DATABASE_NAME = "fasesLunares";

    // Nombre de la tabla
    private static final String TABLE_FASES_LUNARES = "t_fases_lunares";

    // Nombre de las columnas
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "nombre";
    private static final String KEY_ALT_NAME = "nombre_alternativo";
    private static final String KEY_DESCRIPTION = "descripcion";
    private static final String KEY_IMG_RESOURCE = "img_resource";


	public DatabaseHandler(Context context) {
		super(context, DATABASE_NAME, null, DATABASE_VERSION);
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
		String CREATE_TABLE = "CREATE TABLE " + TABLE_FASES_LUNARES + "("
		  + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
		  + KEY_ALT_NAME + " TEXT," + KEY_DESCRIPTION + " TEXT," +
KEY_IMG_RESOURCE + " INT" + ")";
		db.execSQL(CREATE_TABLE);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
		// Borramos la tabla antigua
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_FASES_LUNARES);

		// Volvemos a crearla
		onCreate(db);
	}

}
```

Fragmento de código 26. Extensión de SQLiteOpenHelper

A este código le haría falta métodos para agregar y obtener objetos. Dejamos esta parte para que cada alumno la implemente.

• 4.5.3 Conclusiones sobre la práctica

Es importante destacar que en una aplicación de mediana complejidad es muy probable que tengamos que almacenar datos en la forma de SharedPreferences o en una base de datos SQLite. Incluso el almacenamiento de archivos para tener una caché de imágenes puede ser muy útil en un momento dado.

Si elegimos bien el modelo de datos, abstraerlo hasta una tabla de una base de datos es sencillo. Es por eso por lo que hay que analizar para extraer bien los modelos de datos.

4.6 Práctica 5. Procesamiento de información XML/JSON

En una aplicación Android es muy común que parte del contenido esté en un servidor remoto y haya que obtenerlo mediante peticiones HTTP como ya hemos visto en prácticas anteriores. Lo normal es que esa información que descarguemos venga codificada en XML o en JSON.

Android ofrece herramientas para procesar esta información. Para XML posee nativamente librerías de DOM, SAX y XPATH. Para JSON tiene una librería propia, aunque suele compensar usar la librería externa GSON de Google, por motivos de velocidad de proceso.

• 4.6.1 Parseo de un archivo XML

En este apartado se pide que el alumno implemente una aplicación en la que se procese el XML que se adjunta. Se recomienda copiar los archivos XML y JSON al directorio assets del proyecto Android. Recomendamos estudiar la clase Context para aprender a conseguir un InputStream desde un archivo que se encuentra en la carpeta assets, y poder trabajar con él.

Para el procesado del archivo XML cada alumno puede elegir el mecanismo que prefiera. Los objetos de este XML tienen la siguiente estructura:

```
<CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
</CD>
```

Fragmento de código 27. Objeto XML a procesar

Una vez procesados deberá mostrarse una lista con estos datos. Aquí se muestra una captura de una posible implementación:

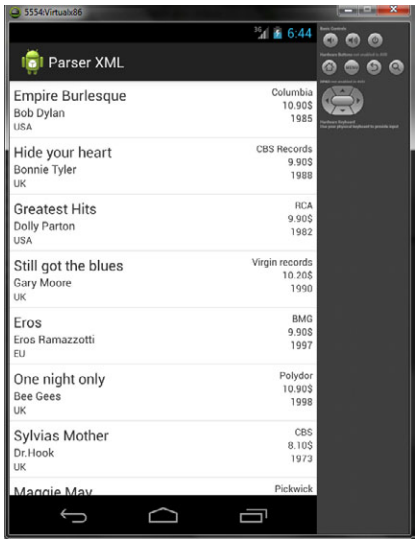


Ilustración 52. Procesamiento XML

Se pide que el alumno calcule el tiempo de proceso con la opción System.currentTimeMillis(), para mostrar por pantalla en un Toast y comparar el tiempo de proceso con el JSON.

• 4.6.2 Parseo de un archivo JSON

En este caso el origen de los datos es un archivo json adjunto al documento, y para procesarlo se puede usar la librería nativa de Android o la clase Gson de Google.

En este caso también se pide mostrar estos datos en una ListView y medir el tiempo usado en el proceso de los archivos.

• 4.6.3 Conclusiones sobre la diferencia entre tiempos de proceso

Se pide que en este caso el alumno realice pruebas empíricas para determinar un valor medio de proceso del archivo XML y del JSON, ambos con mismo contenido.

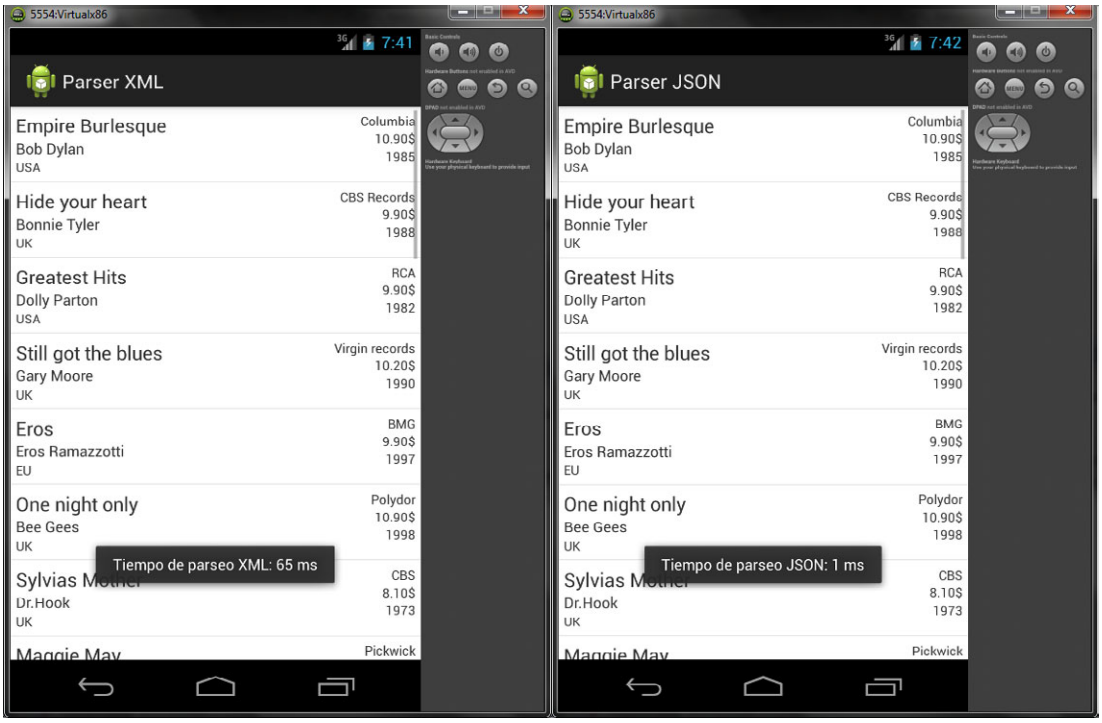


Ilustración 53. Diferencia entre tiempos de procesamiento

• 4.6.4 Anexo práctica 5. Archivos para parsear

Aquí están los archivos necesarios para realizar esta práctica:


cd_catalog.xml


cd_catalog.json

4.7 Práctica 6. Uso de MAPAS Google Maps v2

En esta práctica se abordará una de las características más atractivas del sistema operativo Android, y es el uso de Google Maps.

Google proporciona una librería para tener acceso a las clases necesarias y a los servicios para acceder a los datos de los mapas. Esta librería se puede instalar desde el Android SDK Manager, como puede verse en la siguiente captura:

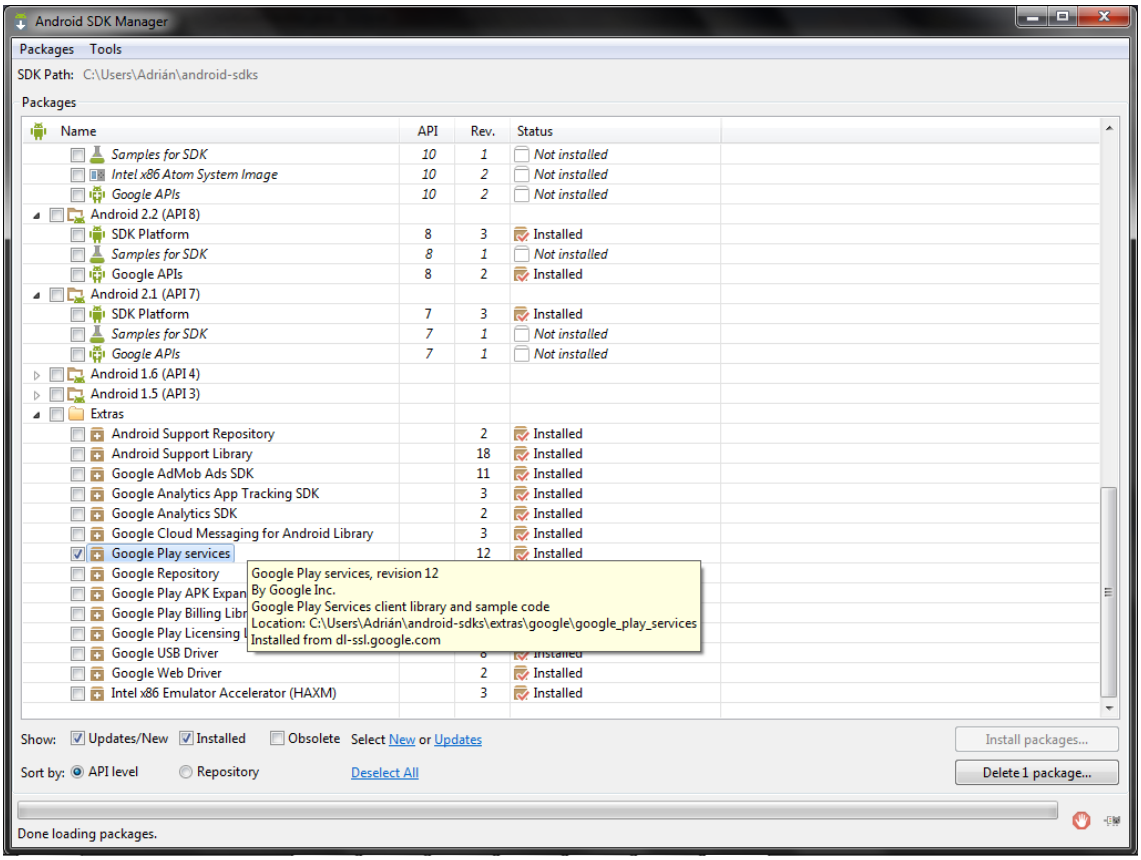


Ilustración 54. Instalación de Google Play Services

Una vez la tengamos instalada, hay que importarla en el workspace. Para ello elegimos la opción File → Import → Android → Existing Android code into Workspace. La librería se encuentra en siguiente path: {path del SDK}\extras\google\google_play_services\libproject.

Una vez importada en el workspace tenemos que hacer que el proyecto la utilice. Para ello vamos a las propiedades del mismo y en la categoría Android, abajo a la derecha elegimos la opción Library → Add y elegimos el proyecto de la librería, que debería llamarse google-play-services-lib.

• 4.7.1 Obtención de un API Key

Para que Google autorice el acceso a sus servidores, hay que registrar la aplicación en la consola de APIs. Es un servicio gratuito para el que únicamente hace falta una cuenta de google. Para registrar la aplicación debemos acceder a esta dirección: https://code.google.com/apis/console. En el menú de la izquierda, en el desplegable, elegimos Create. Una vez lo creado el proyecto, pasaremos a elegir los servicios que queremos tener activos. En este caso es Google Maps Android API v2.

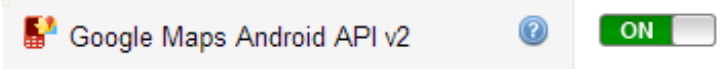


Ilustración 55. Activación Google Maps Api Android v2

Una vez activado, en el menú izquierdo nuevamente hay que ir a la opción API Access. Elegimos la opción Create new Android key. Para crear la API key nos hacen falta dos datos, el nombre del

paquete de la aplicación que se está creando, y el fingerprint SHA1 del certificado de debug de Android. Este último dato se puede sacar directamente de Eclipse, está en Window → Preferences → Android → Build → SHA1 Fingerprint. Una vez se tengan estos datos, pueden ser añadidos tal y como se muestra en la siguiente captura:

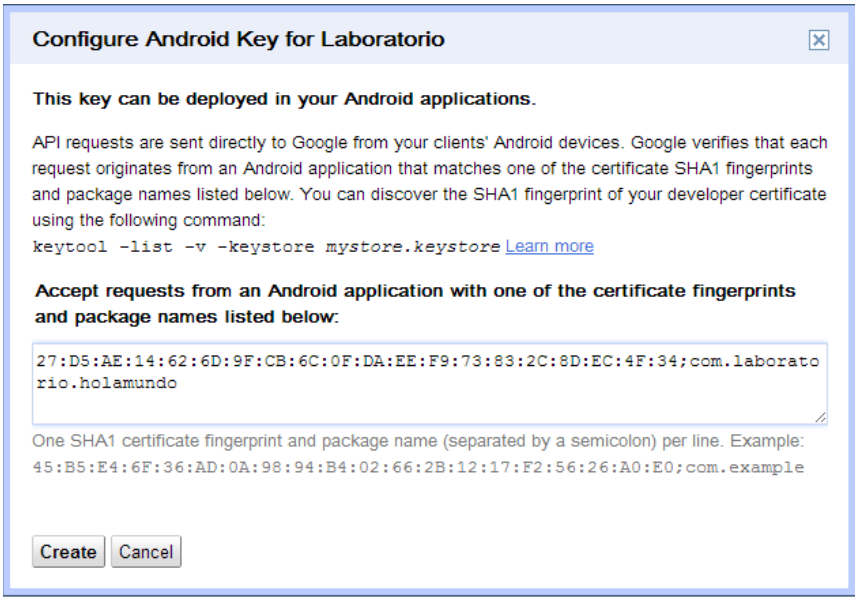


Ilustración 56. Obtención API key Google Maps v2

Una vez se pulse a create, ya se tiene el API key que se necesitará incluir en la aplicación para que tenga acceso a Google Maps.

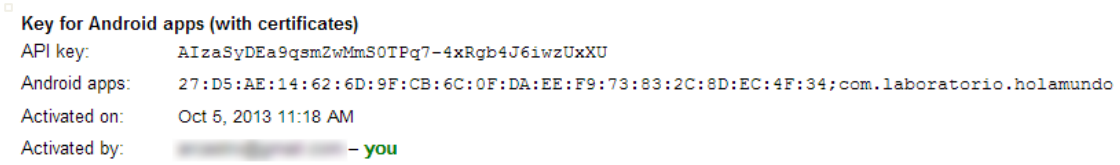


Ilustración 57. Clave API Key v2

• 4.7.2 Permisos en el Android Manifest

Ahora en el Android Manifest hay que añadir una serie de permisos tal y como se indica en este documento de Google:

https://developers.google.com/maps/documentation/android/start#installing_the_google_maps_android_v2_api

El manifest quedará parecido a este:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.laboratorio.holamundo"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyDEa9qsmZwMmS0TPq7-4xRgb4J6iwzUxXU" />

        <activity

android:name="com.laboratorio.holamundo.activities.NavigationDrawerActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity

android:name="com.laboratorio.holamundo.activities.DetalleFaseLunarActivity"
            android:label="@string/detalle" />
    </application>

</manifest>
```

Fragmento de código 28. AndroidManifest para Google Maps

• 4.7.3 Creación de un MapFragment

Una vez se tengan todos los permisos en el manifest ya se puede instanciar el nuevo fragmento que extienda a MapFragment. Para obtener el objeto GoogleMap para poder trabajar con él, en el método onResume hay que llamar al método getMap().

Se pide que el alumno implemente este código y añada Markers a su elección. Estos Markers, cuando sean pulsados deberán mostrar un popup emergente con los datos que se deseen mostrar.

A modo opcional los alumnos podrán dibujar polígonos, polylines, círculos, etc sobre el mapa.

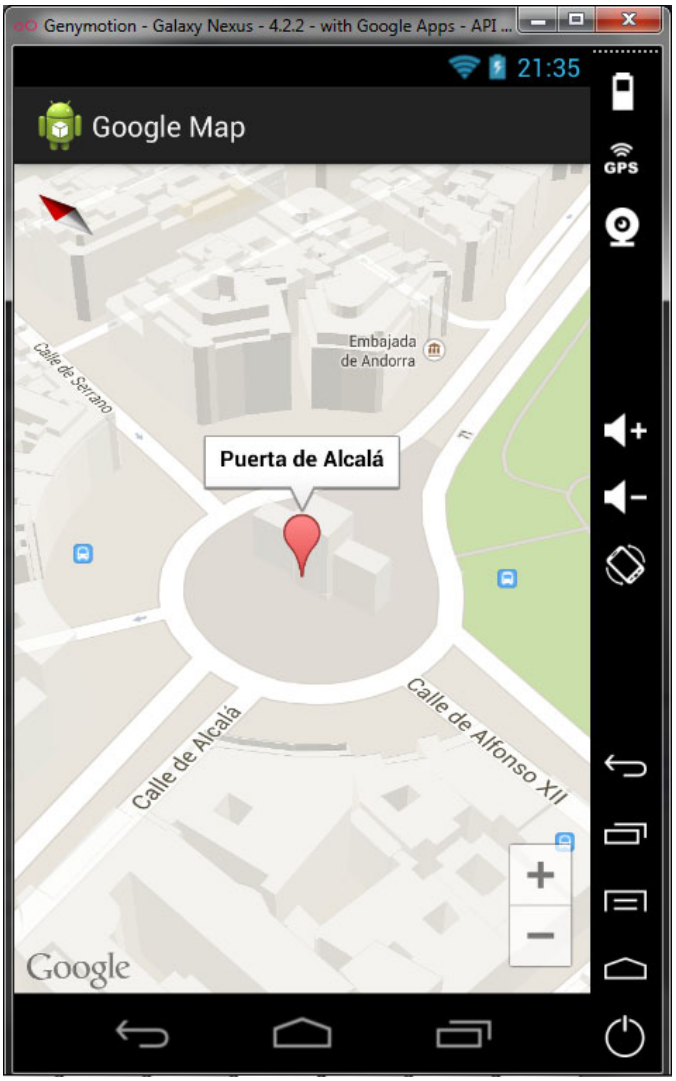


Ilustración 58. MapFragment

4.8 Práctica opcional. Cámaras de tráfico geolocalizadas

En esta práctica se pide que el alumno, utilizando el conocimiento adquirido en las prácticas anteriores desarrolle una aplicación que muestre una lista con las cámaras de tráfico de la comunidad de Madrid.

La aplicación ha de cumplir los siguientes requisitos técnicos:

- Ha de obtener los datos desde la siguiente URL: http://informo.madrid.es/informo/tmadrid/CCTV.kml
- Se han de usar AsyncTask para la cualquier obtención de datos desde internet
- La lista principal de la aplicación ha de tener imágenes en miniatura de la cámara.
- Cuando se pulse en cada uno de los ítems de la lista principal, se ha de mostrar otra Actividad o Fragmento que muestre la siguiente información:
    - Un mapa sobre el que posicionar la cámara.
    - La imagen actual de la cámara.
    - Descripción de la localización de la cámara.

Aquí se muestran unas capturas a modo de ejemplo:

Lista principal:

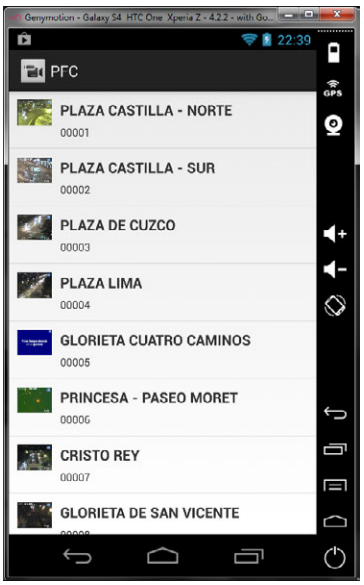


Ilustración 59. Lista de cámaras

Vista de detalle:

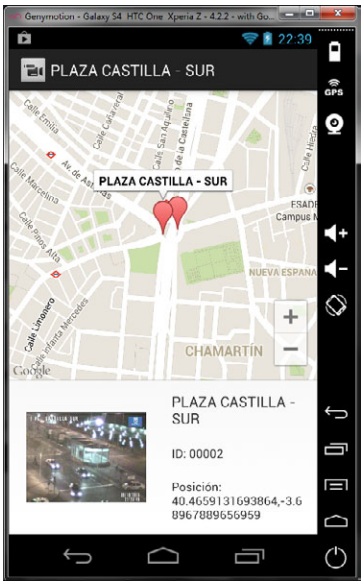


Ilustración 60. Detalle de cámara

CAPÍTULO 5. APLICACIÓN CÁMARAS DE TRÁFICO DE LA CAM

En este último capítulo se explicarán los detalles de la aplicación que se ha usado como ejemplo de la última lección del laboratorio.

5.1 Diseño de la aplicación

La aplicación "Cámaras de tráfico de la CAM" es una aplicación que permite al usuario acceder a información georeferenciada del estado del tráfico en la ciudad de Madrid. Para ello se usará la información proveniente de http://informo.munimadrid.es/. En dichos servicios se puede encontrar el siguiente archivo http://informo.munimadrid.es/informo/tmadrid/CCTV.kml.

Es un archivo kml contiene información de las cámaras que se podrían sintetizar en el siguiente modelo de datos:

Atributo	Tipo	Descripción
cameraId	String	Identificador único de cada cámara
latitude	Double	Latitud en la que está la cámara
longitude	Double	Longitud en la que está la cámara
name	String	Descripción de la cámara
image	String	URL en la que se encuentra la imagen de la cámara

Tabla 9. Modelo de datos Camera

La aplicación va a constar de dos pantallas que se explicarán a continuación.

• 5.1.1 Pantalla principal. Listado de cámaras

En esta pantalla principal el usuario podrá ver la totalidad de cámaras disponibles, ordenadas por su identificador, previsualizando cada una de las cámaras.

- Requisitos funcionales asociados a la pantalla.
  - La información ha de descargarse desde internet sin afectar a la usabilidad de la aplicación para ser mostrada posteriormente en un ListView.
  - Cada una de las cámaras deberá tener una previsualización en cada elemento de la lista.
  - La información que ha de mostrar cada uno de los ítems de la lista es el identificador de la cámara y su descripción, a parte de la previsualización.

- Requisitos no funcionales asociados a la pantalla.
  - Cuando el usuario haga scroll, la aplicación deberá mantener un grado alto de usabilidad y deberá hacer un uso eficiente de los recursos, ya sean datos, memoria, o cpu.
  - Si el usuario no está conectado a internet, la pantalla no deberá fallar y avisará al usuario indicándole el problema.
  - Si ha habido algún error de descarga, la pantalla no deberá fallar y avisará al usuario indicándole el problema.
  - La pantalla deberá tener soporte de orientaciones.

La pantalla en cuestión está representada en la siguiente captura de pantalla.

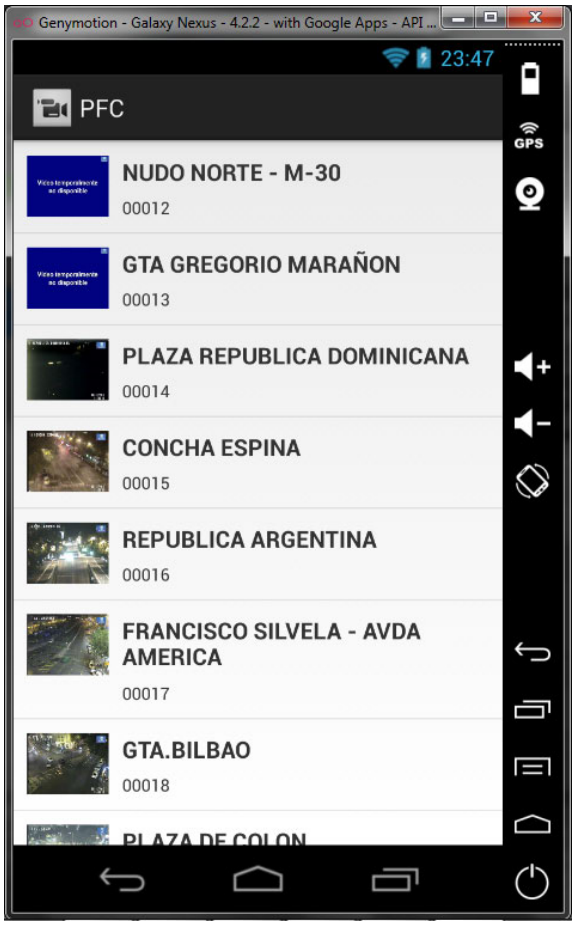


Ilustración 61. Lista de cámaras

Ilustración 62. Lista de cámaras landscape

Se ha creado para la descarga del archivo kml un *parser* DOM que devuelve un ArrayList de objetos Camera para poder poblar la lista. Cada vez que se instancia cada uno de los elementos de la lista y el usuario no está haciendo scroll, se lanza un proceso asíncrono para descargar la imagen sin bloquear la interfaz de usuario. Para ello he desarrollado una clase de ayuda que descarga la imagen y devuelve un mensaje por medio de una interfaz si la descarga ha sido correcta o por el contrario ha habido un error.

La forma de utilizarla desde el hilo principal de la aplicación es dentro del Adapter que puebla la lista, de esta manera:

```
mImDownloader.ImageLoader(mCamera.getImage(),
        holder.iv, true, new ImageBackgroundLoadingListener() {

        @Override
        public void onError(ImageView mImageView, String motivo) {
                mImageView.setImageResource(R.drawable.ic_launcher);
        }

        @Override
        public void onCompleted(ImageView mImageView, Bitmap mBitmap) {
                mImageView.setImageBitmap(mBitmap);
        }
});
```

Fragmento de código 29. ImageLoader

Así lo que se ha conseguido es que la carga de la imagen se haga de manera independiente y eficaz.

Otro aspecto importante para optimizar el uso de recursos es cuando el usuario hace scroll en la lista. Ya se ha explicado en la práctica 1 cómo se implementa el patrón ViewHolder para evitar instanciar más objetos de los que se ven en una ListView, sin embargo si no bloqueáramos la descarga de la imagen cuando el usuario hace scroll de manera rápida, podríamos obtener un ANR (Application Not Responding). El motivo es que el adapter intentaría dibujar esa imagen descargándola de internet lanzando para ello un AsyncTask nuevo. Sería posible que se lanzaran tantos AsyncTasks, que el sistema se quedara sin memoria.

Para evitar este problema, en la lista se implementó el listener para cuando el usuario hace scroll, y se bloqueó la descarga de imágenes mientras la lista está en movimiento. Cuando el scroll para, las imágenes de esa pantalla se descargan automáticamente.

A continuación está el código que se ocupa de dotar de este funcionamiento a la lista.

```
/**
 * Añadimos un onScrollListener para evitar que el adapter haga una carga innecesaria de imágenes desde la web.
 * De esta manera solo carga imágenes cuando el scroll está en IDLE.
 */
mListView.setOnScrollListener(new OnScrollListener() {

	@Override
	public void onScrollStateChanged(AbsListView view, int scrollState) {
		if (scrollState == AbsListView.OnScrollListener.SCROLL_STATE_IDLE) {
			mAdapter.isListViewScrolling(false);
			//para forzar el redibujado
			mAdapter.notifyDataSetChanged();
		}
		else {
			mAdapter.isListViewScrolling(true);
		}
	}

	@Override
	public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount) {}
});
```

Fragmento de código 30. ScrollListener

• 5.1.2 Pantalla detalle. Información extendida y posición de la cámara

En esta pantalla, lo que se le muestra al usuario es un mapa geolocalizando las cámaras de tráfico y se muestra información cada vez que el usuario pulsa en una nueva cámara.

- Requisitos funcionales asociados a esta pantalla
  - La pantalla deberá mostrar un mapa con un marcador sobre cada una de las posiciones de las cámaras descargadas.
  - Cuando el usuario pulse sobre uno de estos marcadores, aparecerá una ventana de tipo pop-up indicando la descripción de la cámara. Al mismo tiempo la vista de detalle a pie del mapa se actualizará mostrando la imagen de la cámara pulsada
- Requisitos no funcionales asociados a esta pantalla.
  - La pantalla deberá tener soporte a diferentes orientaciones.
  - Si el usuario no posee Google Play Services, la aplicación deberá avisarle para que actualice su sistema.

A continuación se muestran unas capturas de esta pantalla en las orientaciones soportadas.

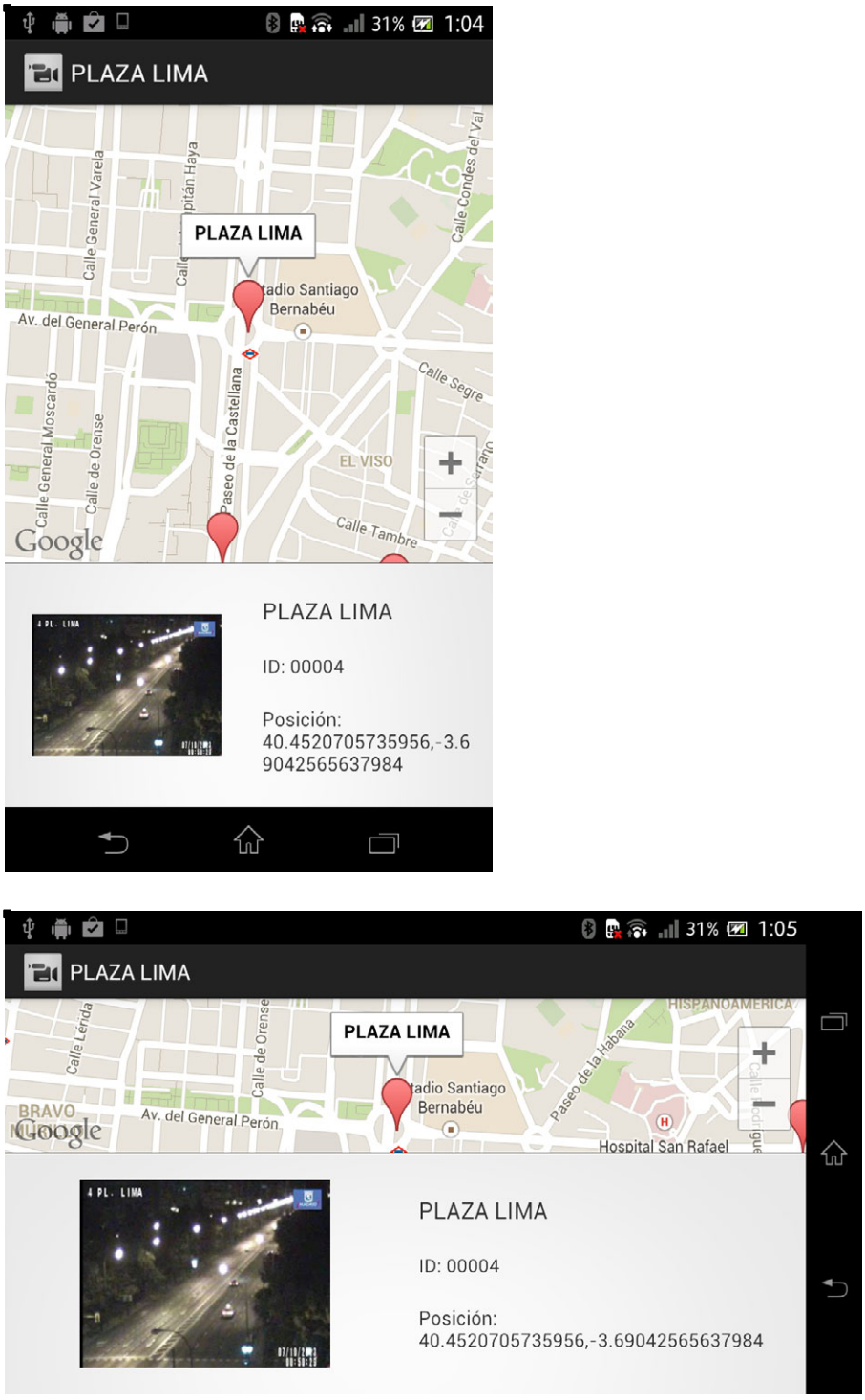


Ilustración 63. Vista detalle de cámara. Portrait y Landscape

Si el usuario hace click sobre la imagen que se le presenta, se pasará a mostrar la imagen a pantalla completa, con soporte de orientaciones.

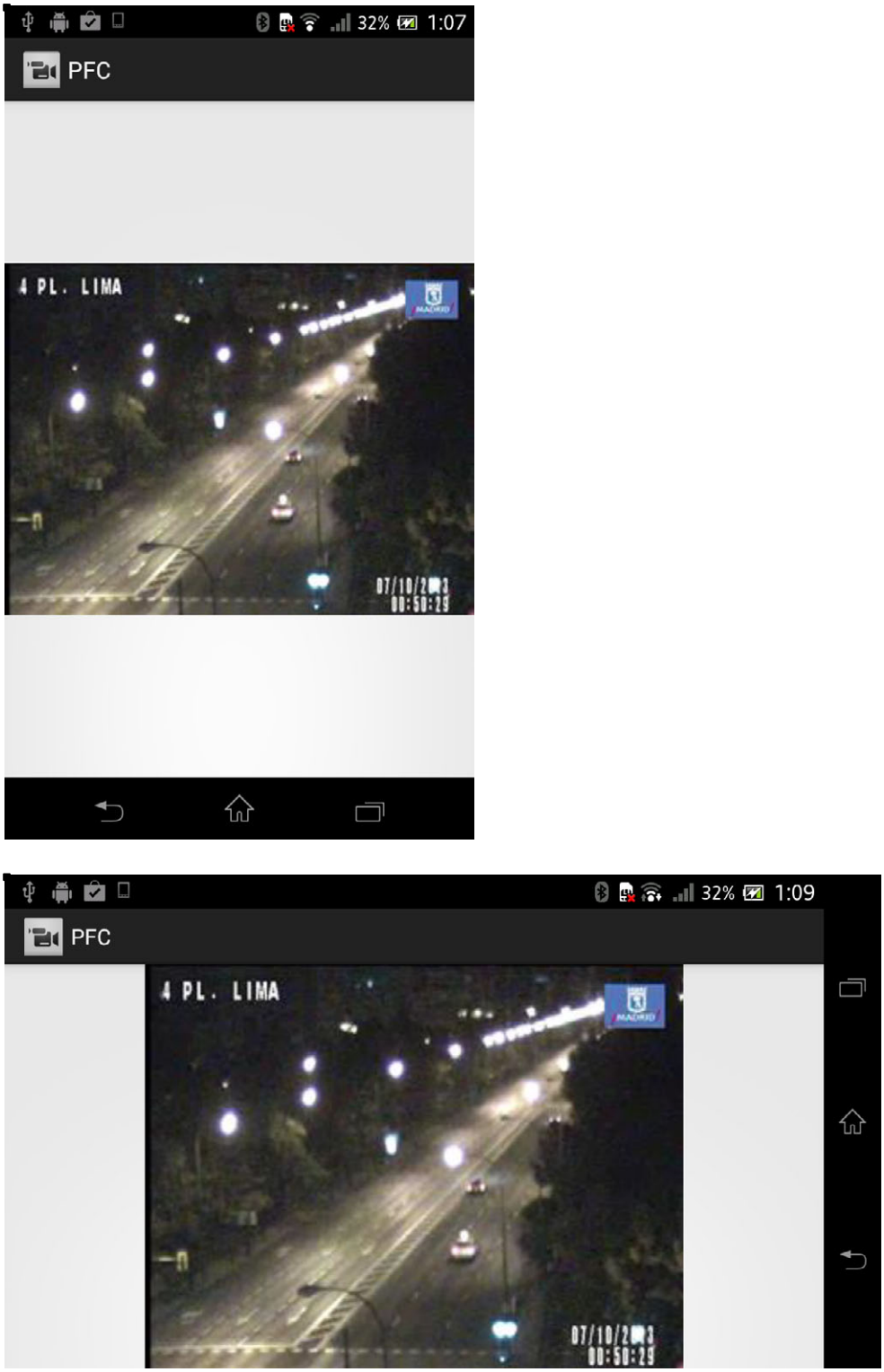


Ilustración 64. Vista de cámara pantalla completa. Portrait y Landscape

CAPÍTULO 6. CONCLUSIONES Y TRABAJOS FUTUROS

Este capítulo aglutina las conclusiones obtenidas tras realizar este Proyecto Fin de Carrera así como los trabajos futuros que pueden verse derivados desde el mismo.

6.1 Conclusiones

El objetivo principal de este Proyecto Fin de Carrera era la creación de una serie de prácticas incrementales con el fin de que un alumno pudiera obtener la base suficiente para emprender un reto profesional en este ámbito.

Se realizó un estudio del estado del arte de los diferentes sistemas operativos actuales del mercado. Como ha podido verse, el incremento en el uso de terminales con sistema operativo Android ha sido notable y la demanda de aplicaciones y soluciones para esta plataforma se ha visto afectada también (www.tecnologiapyme.com, 2013).

En la actualidad la demanda laboral de personas con conocimientos técnicos avanzados de movilidad ha aumentado y las empresas buscan gente con este perfil (Universia España, 2013). Es por esto que asignaturas que proporcionen una base en este ámbito, son muy necesarias para que los alumnos puedan en un futuro orientarse hacia una profesión que les abra puertas en el mercado laboral europeo.

Las lecciones han sido concebidas para que un alumno consiga una base de buenas prácticas y un conocimiento extenso de la arquitectura de componentes del sistema operativo Android. La lección final bien puede ser una aplicación real disponible en el Google Play.

Es necesario recalcar que las capacidades de proceso y de memoria de los móviles crece día a día, pero lo más complicado de realizar con éxito una aplicación es mantener unos criterios de sencillez y optimización de código que permitan que dicha aplicación pueda correr en terminales de gama baja.

6.2 Trabajos futuros

Las lecciones presentadas en este Proyecto Fin de Carrera abarcan la gran mayoría de componentes básicos del sistema operativo Android. Sin embargo, y como es lógico, cada día existen nuevas actualizaciones del sistema operativo, nuevas APIs, y nuevas demandas del mercado en cuanto a funcionalidades o apariencia.

Es por esto que las siguientes lecciones pueden presentarse como mejoras y/o trabajos futuros a este Proyecto Fin de Carrera:

- **Cloud Computing**: Google ofrece a los usuarios gran diversidad de aplicaciones en la nube (Google Inc., 2014), como puede ser Google Drive, Google Docs, Google Calendar, etc. Todas ellas tienen posibilidades de integración con aplicaciones nativas y ofrecen innumerables posibilidades de interconexión.
- **Xml Pull Parser**: En las prácticas referentes al tratamiento de XML y JSON se ha hablado de parseadores típicos como pueden ser SAX o DOM. XML Pull Parser viene de base en las APIs disponibles en Android, ofrece gran facilidad de uso al desarrollador y es más liviano que los antes mencionados.
- **Creación de la misma pantalla para diferentes dispositivos:** Uno de los problemas mayores de Android es la fragmentación de dispositivos. Podemos encontrar dispositivos con resoluciones dispares, por lo que muchas veces una misma pantalla ha de ser presentada de manera diferente según el dispositivo.
- **Uso de la librería de soporte v4:** Según estadísticas de Google (Google Inc., 2014), casi el 20% de los dispositivos presentes en el Google Play son de versiones anteriores a la 4.0.3 (Jelly Bean). Muchos de los componentes introducidos en esta versión (Fragments, ActionBar, LocalBroadcastManager, etc) ofrecen facilidades e incluso son de uso recomendado según las guías de diseño de Android, por lo que el conocimiento del uso de esta librería de soporte es un buen añadido.
- **Animaciones:** Cada vez más los clientes demandan aplicaciones que cumplan con los requisitos funcionales, pero que sean atractivas a la vista.

BIBLIOGRAFÍA

Europa Press. (21 de agosto de 2013). *www.abc.es*. Recuperado el 15 de octubre de 2013, de http://www.abc.es/tecnologia/moviles-telefonia/20130820/abci-espana-lider-penetracion-smartphone-201308201842.html

Google Inc. (2013). *developer.android.com/guide/components/activities.html*. Recuperado el 19 de noviembre de 2013, de http://developer.android.com/guide/components/activities.html

Google Inc. (2013). *developer.android.com/guide/components/fragments.html*. Recuperado el 15 de noviembre de 2013, de http://developer.android.com/guide/components/fragments.html

Google Inc. (2013). *developer.android.com/guide/components/services.html*. Recuperado el 12 de diciembre de 2013, de http://developer.android.com/guide/components/services.html

Google Inc. (2014). *code.google.com*. Recuperado el 1 de abril de 2014, de https://code.google.com/p/google-api-java-client/wiki/APIs

Google Inc. (2014). *developer.android.com*. Recuperado el 18 de marzo de 2014, de http://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net

GSMArena. (2008). *www.gsmarena.com*. Recuperado el 5 de noviembre de 2013, de http://www.gsmarena.com/t_mobile_g1-2533.php

Lount, P. W. (16 de agosto de 2004). *www.smalltalk.org*. Recuperado el 17 de octubre de 2013, de http://www.smalltalk.org/smalltalk/whatissmalltalk.html

Meier, R. (2010). *Professional Android 2 Application Development.* Indianapolis: Wiley Publishing, Inc.

Meier, R. (2012). *Professional Android 4 Application Development.* Indianapolis: John Wiley & Sons, Inc.

Open Handset Alliance. (5 de noviembre de 2007). *www.openhandsetalliance.com*. Recuperado el 2 de noviembre de 2013, de http://www.openhandsetalliance.com/press_110507.html

TIOBE. (s.f.). *www.tiobe.com*. Recuperado el 10 de octubre de 2013, de http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Universia España. (3 de noviembre de 2013). *www.universia.es*. Recuperado el 2 de febrero de 2014, de http://noticias.universia.es/empleo/noticia/2013/03/11/1009922/desarrolladores-aplicaciones-moviles-son-mas-demandados-industria-tecnologica.html

www.tecnologiapyme.com. (12 de diciembre de 2013). Recuperado el 20 de enero de 2014, de http://www.tecnologiapyme.com/movilidad/crece-el-desarrollo-de-apps-corporativas-android